UNIVERSITATEA TEHNICĂ DE CONSTRUCȚII

BUCUREŞTI

BAZELE INFORMATICII

SI

LIMBAJE DE PROGRAMARE

2

BAZE DE DATE RELAȚIONALE

Romică TRANDAFIR

Mihai NISTORESCU **Ion MIERLUŞ-MAZILU**

Bucureşti

2007

Cuprins

Cuprins **1**
1. Bănci şi baze de date **5**
1.1. Noţiuni generale 5
1.2. Sisteme de baze de date 6
1.3. Organizarea datelor într-o bază de date 7
1.4. Modelarea la nivel logic a datelor într-o bază de date 8
1.5. Sistem de gestiune a bazelor de date 10
1.5.1. Interfeţe SGBD 11
1.5.2. Funcţii şi servicii oferite de un SGBD 12
1.5.3. Activităţile asigurate de SGBD 14
1.5.4. Obiectivele unui SGBD 15
2. Etapele de realizare a unei bănci de date **17**
2.1. Etapa de analiză 17
2.2. Etapa de programare 17
2.3. Punerea în funcţiune şi exploatarea bazei de date 18
2.4. Documentaţia aplicaţiei 19
3. Sisteme de gestiune a bazelor de date relaţionale **20**
3.1. Modelul relaţional al datelor 20
3.1.1. Structura relaţională a datelor 20
3.1.2. Operaţii în algebra relaţională 21
3.1.3. Optimizarea cererilor de date 28
3.1.4. Baze de date relaţionale 30
3.1.5. Tipuri de relaţii 31
3.1.6. Restricţia de cardinalitate 33
3.2. Regulile lui Codd 34
3.3. Normalizarea datelor 37
3.4. Cerinţele minimale de definire a SGBDR 37
4. Componentele funcţionale ale sistemului Visual FoxPro **39**
4.1. Programarea orientată pe obiecte 39
4.2. Ierarhia claselor în Visual FoxPro 44
4.2. Arhitectura VFP 6.0 45
4.3. Moduri de lucru în VFP 46
4.3.1. Modul de lucru meniu sistem 47
4.3.2. Modul de lucru prin comenzi 52
5. Organizarea datelor înVisual FoxPro **53**
5.1. Manipularea bazei de date şi a tabelelor în VFP 53
5.1.1. Crearea bazei de date 53
5.1.2. Crearea tabelelor 54
5.1.3. Validarea câmpurilor unei înregistrări la introducere 55
5.2. Deschiderea bazei de date/tabelei 61
5.3. Consultarea şi modificarea bazei de date/tabelei 62
5.3.1. Modificarea structurii de date a tabelelor în modul asistat 62
5.3.2. Deplasări în tabelă. Căutări secvenţiale 64
5.4. Închiderea bazei de date/tabelei 65
6. Utilizarea asistentului Wizard în VFP **67**
6.1. Table Wizard 67
6.2. Form Wizard 68
6.3. Report Wizard 69

6.4. Label Wizard 70
6.5. Query Wizard 70
6.6. Mail Merge Wizard 72
6.7. Editorul de texte în VFP 6.0 73
6.7.1. Lansarea editorului 73
6.7.2. Lucrul cu blocul de linii din fişier 73
7. Elemente ale limbajului propriu Visual FoxPro 6.0 74
7.1. Simboluri 74
7.2. Variabile de sistem 74
7.3. Comentariul 74
7.4. Tipuri de date, operaţii şi funcţii 74
7.4.1. Date de tip numeric. Funcţii aritmetice şi financiare 77
7.4.2. Aplicaţii ale funcţiilor financiare 78
7.4.3. Date şi funcţii de tip caracter 81
7.4.4. Date şi funcţii de tip calendaristic 83
7.4.5. Date de tip memo 84
7.5. Variabile şi masive 84
7.5.1. Variabile 84
7.5.2. Macrosubstituţia 85
7.5.3. Masive de date 86
7.6. Funcţii de prelucrare a masivelor 87
8. Accesul şi actualizarea datelor 92
8.1. Manipularea structurii unei tabele 92
8.2. Ordonarea datelor 93
8.3. Accesul la date 98
8.4. Actualizarea datelor 103
8.4.1. Adăgarea de noi înregistrări 103
8.4.2. Modificarea înregistrărilor 106
8.4.3. Ştergerea înregistrărilor 110
8.4.5. Relaţii între tabele 113
9. Programarea procedurală 120
9.1. Programarea structurată 120
9.1.1. Structura liniară 121
9.1.2. Structura alternativă 124
9.1.3. Structura repetitivă 126
9.2. Modularizarea programelor 129
9.2.1. Proceduri 130
9.2.2. Funcţii 131
10. Comenzi ale nucleului SQL 135
11. Proiectarea meniurilor şi a barelor de instrumente 146
12. Aplicaţii 147
12.1. Evidenţa rezultatelor activităţii studenţilor într-o facultate 147
12.1.1. Formularea şi analiza problemei 147
12.1.2. Crearea, actualizarea, modificarea şi interogarea tabelelor 149
12.1.3. Crearea structurii unui fişier 150
12.1.4. Adresarea prin macrosubstituţie 151
12.1.5. Crearea unui meniu 151
12.2. Gestionarea unei magazii 153
12.2.1. Proiectarea şi realizarea aplicaţiei 153
12.2.2. Realizarea documentaţiei şi a aplicaţiei în format executabil 220

12.2.3. Utilizarea Setup Wizard 225
A. Anexa 227
Index de termeni 229
Bibliografie 231

1. Bănci şi baze de date

1.1. Noțiuni generale

O *bază de date* reprezintă un ansamblu de date integrat, anume structurat şi dotat cu o descriere a acestei structuri. Descrierea structurii poartă numele de *dicționar de date* sau *metadate* şi crează o interdependență între datele propriu-zise şi programe.

Baza de date poate fi privită ca o colecție de fişiere interconectate care conțin nucleul de date necesare unui sistem informatic. Astfel, poate fi considerată drept un model al unor aspecte ale realității unei unități economice, modelată prin intermediul datelor. Diferitele obiecte din cadrul realității, ce prezintă interes, sunt denumite *clase* sau *entități*. Pentru aceste obiecte sunt achiziționate şi memorate date referitoare la diferite *caracteristici* (*atribute*). Baza de date se constituie ca un ansamblu intercorelat de colecții de date, prin care se realizează reprezentarea unei realități.

Datele constituie orice mesaj primit de un receptor, sub o anumită formă.

Informațiile reprezintă cantitatea de noutate adusă de un mesaj din exterior (realitate).

Un *fişier* este un ansamblu de *înregistrări fizice*, omogene din punct de vedere al conținutului şi al prelucrării.

O *înregistrare fizică* este o unitate de transfer între memoria internă şi cea externă a calculatorului.

O *înregistrare logică* este unitatea de prelucrare din punct de vedere al programului utilizator.

O înregistrare se compune din *câmpuri* (*atribute*) care descriu anumite aspecte ale realității. Câmpurile sunt înregistrări logice.

O bază de date trebuie să asigure:

- *abstractizarea datelor* (baza de date fiind un model al realității),

- *integrarea datelor* (baza de date este un ansamblu de colecţii de date intercorelate, cu redundanţă controlată),
- *integritatea datelor* (se referă la corectitudinea datelor încărcate şi manipulate astfel încât să se respecte restricţiile de integritate),
- *securitatea datelor* (limitarea accesului la baza de date),
- *partajarea datelor* (datele pot fi accesate de mai mulţi utilizatori, eventual în acelaşi timp),
- *independenţa datelor* (organizarea datelor să fie transparentă pentru utilizatori, modificările în baza de date să nu afecteze programele de aplicaţii).

1.2. Sisteme de baze de date

Sistemele de baze de date, sau *băncile de date,* reprezintă un sistem de organizare şi prelucrare, respectiv teleprelucrare (prelucrare la distanţă) a informaţiei, constituit din următoarele trei elemente:

- colecţie de date aflate în interdependenţă
- descrierea datelor şi a relaţiilor dintre ele
- un sistem de programe care asigură exploatarea bazei de date (actualizare, interogare).

creare baza de date

Arhitectura sistemului de baze de date este formată din următoarele componente (Figura 1):

- baza/bazele de date – reprezintă componenta de tip date a sistemului (colecţiile de date propriu-zise, indecşii);
- sistemul de gestiune a bazei/bazelor de date – ansamblul de programe prin care se asigură gestionarea şi prelucrarea complexă a datelor şi care reprezintă componenta software a sistemului de baze de date (Sistem de Gestiune a Bazelor de Date – SGBD);
- alte componente – proceduri manuale sau automate, inclusiv reglementări administrative, destinate bunei funcţionări a sistemului, dicţionarul bazei de date (metabaza de date) care conţine informaţii despre date, structura acestora, elemente de descriere a semanticii, statistici, documentaţii, mijloacele hardware utilizate, personalul implicat.

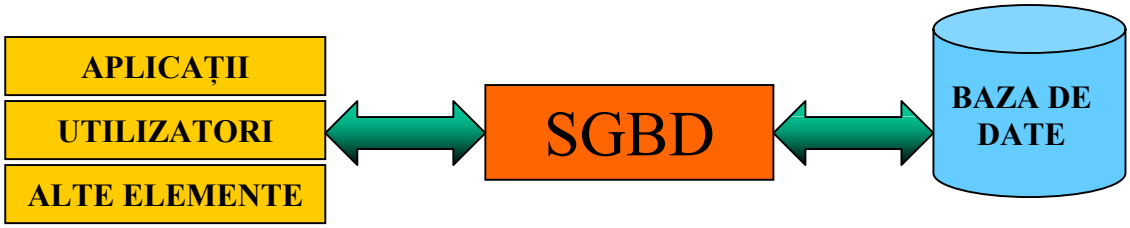


Fig. 1

1.3. Organizarea datelor într-o bază de date

Arhitectura internă a unui sistem de baze de date conform standardului ANSI/X3/SPARC (1975) conține trei niveluri funcționale. O caracteristică fundamentală a bazelor de date este aceea că produce câteva niveluri de abstractizare a datelor prin ascunderea (transparența) detaliilor legate de stocarea datelor, utilizatorilor.

Se defineşte *modelul datelor*, ca un set de concepte utilizat în descrierea structurii datelor. Prin *structura bazei de date* se înțelege tipul datelor, legătura dintre ele, restricțiile aplicate datelor. O structură de date asociată unei baze de date poate fi reprezentată pe trei niveluri, Figura 2, astfel:

- *Nivelul intern* – constituit din schema internă ce descrie structura de stocare fizică a datelor în baza de date, utilizând un model al datelor fizice. La acest nivel se descriu detaliile complete ale stocării şi modul de acces la date.
- *Nivelul conceptual* – sau schema conceptuală, descrie structura întregii baze de date pentru o cumunitate de utilizatori. La nivel conceptual se face o descriere completă a bazei de date ascunzându-se detaliile legate de stocarea fizică şi detaliind descrierea entităților, tipurilor de date, relațiile dintre ele şi restricțiile asociate.

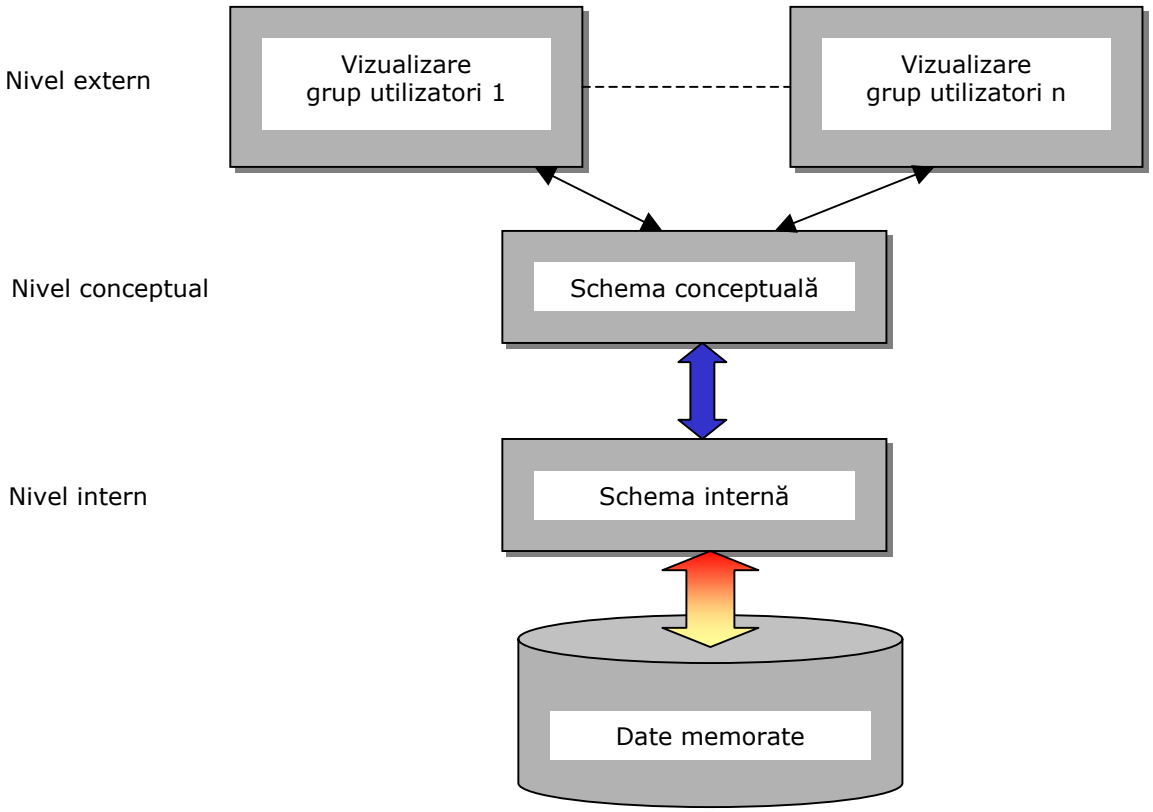


Fig. 2

- *Nivelul extern* – sau nivelul vizual (utilizator), include o colecție de scheme externe ce descriu baze de date prin prisma diferiților utilizatori.

În multe SGBD nu se poate face o distincție clară între cele trei niveluri, de multe ori nivelul conceptual este puternic dezvoltat şi suplineşte aparent celelalte niveluri. De asemenea, la dezvoltarea aplicațiilor se remarcă o contopire a nivelului extern cu cel conceptual.

De la modelul conceptual, cererile sunt adresate modelului intern pentru a fi procesate şi aplicate datelor stocate. Procesul de transfer a cererilor şi rezultatelor între nivele se numeşte *cartografiere* (mapping).

În funcție de categoria de personal implicată în folosirea bazei de date, datele dintr-o bază de date pot fi structurate pe trei niveluri:

- *nivelul conceptual* (global) – exprimă viziunea administratorului bazei de date asupra datelor. Acestui nivel îi corespunde structura conceptuală (schema) a bazei de date, prin care se realizează o descriere a tuturor datelor, într-un mod independent de aplicații, ce face posibilă administrarea datelor.
- *nivelul logic* – exprimă viziunea programatorului de aplicație asupra datelor. La acest nivel se realizează o descriere a datelor corespunzătoare unui anumit program de aplicație.
- *nivelul fizic* – care exprimă viziunea inginerului de sistem asupra datelor. Corespunde schemei interne a bazei de date prin care se realizează o descriere a datelor pe suport fizic de memorie.

1.4. Modelarea la nivel logic a datelor într-o bază de date

Modelul de date reprezintă ansamblul de concepte şi instrumente necesare pentru a construi o schemă a bazei de date. Modelarea datelor poate viza totalitatea datelor din cadrul bazei de date (schema/arhitectura datelor) sau o parte a acestora (subscheme ale bazei de date). Schema şi subschema bazei de date sunt modelele logice ale bazei de date, care au asociate principii generale pentru gestionarea/definirea (structurarea) datelor, manipularea şi asigurarea integrității datelor, fără a reflecta modul de reprezentare şi stocare a acestor date pe suportul de memorie (care sunt ele modelului fizic).

Se cunosc mai multe tipuri de baze de date după modul de organizare, modul de dispunere pe suport magnetic a informației şi a elementelor componente:

- *modele primitive* – datele sunt organizate la nivel logic în fişiere, structura de bază este înregistrarea, mai multe înregistrări fiind grupate în structuri de tip fişier;
- *baze de date ierarhice* – legăturile dintre date sunt ordonate unic, accesul se face numai prin vârful ierarhiei, un subordonat nu poate avea decât un singur superior direct şi nu se poate ajunge la el decât pe o singură cale;
- *baze de date în rețea* – datele sunt reprezentate ca într-o mulțime de ierarhii, în care un membru al ei poate avea oricâți superiori, iar la un subordonat se poate ajunge pe mai multe căi;
- *baze de date relaționale* – structura de bază a datelor este aceea de *relație – tabelă*, limbajul SQL (Structured Query Language) este specializat în comenzi de manipulare la nivel de tabelă. Termenul relațional a fost introdus de un cercetător al firmei IBM, dr. E. F. Codd, în 1969, cel care a enunțat cele 13 reguli de bază necesare pentru definerea unei baze de date relaționale. Baza de date relațională reprezintă o mulțime structurată de date, accesibile prin calculator, care pot satisface în timp minim şi într-o manieră selectivă mai mulți utilizatori. Această mulțime de date modelează un sistem sau un proces din lumea reală şi serveşte ca suport unei aplicații informatice;
- *baze de date distribuite* – sunt rezultatul integrării tehnologiei bazelor de date cu cea a rețelelor de calculatoare. Sunt baze de date logic integrate, dar fizic distribuite pe mai multe sisteme de calcul. Integrarea bazei de date distribuite se face cu ajutorul celor trei tipuri de de scheme care sunt implementate:
    1. *schema globală* – defineşte şi descrie toate informațiile din baza de date distribuită în rețea;
    2. *schema de fragmentare* – descrie legăturile dintre o colecție globală şi fragmentele sale. Ea este de tipul unu la mai mulți şi are forma unei ierarhii;
    3. *schema de alocare* – descrie modul de distribuire a segmentelor pe calculatoarele (nodurile) din rețea. Fiecare segment va avea o alocare fizică pe unul sau mai multe calculatoare. Schema de alocare introduce o redundanță minimă şi controlată: un anumit segment se poate regăsi fizic pe mai multe calculatoare.

Utilizatorul unei asemenea baze de date o vede ca pe o bază de date unică, compactă (nivel logic), cu toate că în realitate ea este distribuită pe mai multe calculatoare legate în rețea (nivel fizic). Această organizare a dus la o creştere substanțială a vitezei de acces la o bază de date într-o rețea de calculatoare. Anumite date stocate pe un server local sunt mult mai rapid accesate decât

dacă ele s-ar afla pe un server la distanță, unde baza de date ar fi fost stocată în întregime (nedistribuită);

- *modele semantice – orientate spre obiecte*. Aceste modele sunt orientate pe reprezentarea semnificației datelor. Structura de bază folosită pentru reprezentarea datelor este cea de clasă de obiecte definită prin abstractizare din entitatea fizică pe care o regăsim în lumea reală. Aici există entități simple şi clase de entități care se reprezintă prin obiecte simple sau clase de obiecte, ordonate în ierarhii de clase şi subclase. Acest tip de bază de date a apărut din necesitatea gestionării obiectelor complexe: texte, grafice, hărți, imagini, sunete (aplicații multimedia) şi a gestionării obiectelor dinamice: programe, simulări.

1.5. Sistem de gestiune a bazelor de date

Sistemele de gestiune a bazalor de date (DBMS – DataBase Management System) sunt sisteme informatice specializate în stocarea şi prelucrarea unui volum mare de date, numărul prelucrărilor fiind relativ mic.

Termenul de bază de date se va referi la datele de prelucrat, la modul de organizare a acestora pe suportul fizic de memorare, iar termenul de gestiune va semnifica totalitatea operațiilor ce se aplică asupra datelor din baza de date.

În arhitectura unui sistem de baze de date SGBD ocupă locul central.

Un SGBD este ansamblul software interpus între utilizatori şi baza de date şi este un interpretor de cereri de acces sau regăsire de date în baza de date, execută cererea şi returnează rezultatul. SGBD este un sistem de programe care facilitează procesul definirii, construcției, organizării şi manipulării datelor pentru diverse aplicații. Utilizatorul are acces la SGBD prin intermediul unei interfețe (aplicație) cu ajutorul căreia stabilesc parametrii interogării şi se primeşte răspuns; întreg ansamblul este descris în Figura 3.

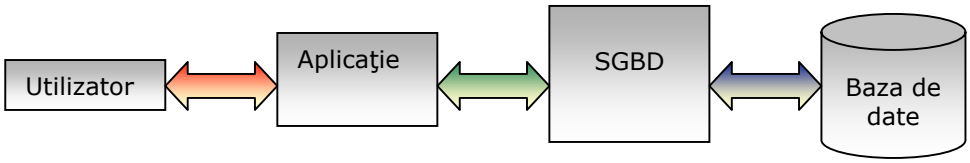


Fig. 3

Organizarea pe trei niveluri a sistemelor de baze de date este strâns legată de conceptul de independență a datelor, în sensul că sistemul bazei de date poate fi modificat la orice nivel fără a

afecta nivelurile superioare. Independența datelor poate fi privită în două moduri, corespunzătoare nivelurilor conceptual (logic) şi intern (fizic).

Independența logică a datelor reprezintă capacitatea modificării schemei conceptuale fără a provoca modificări în schema externă sau în programele de aplicațe. Schema conceptuală se poate modifica prin mărirea bazei de date datorită adăugării de noi tipuri de înregistrări (o nouă coloană într-o tabelă) sau date (înregistrări) sau micşorarea bazei de date în cazul ştergerii unor înregistrări.

Independențafizică a datelor este dată de capacitatea de schimbare a schemei interne fără modificarea schemei conceptuale sau externe.

Funcționarea unui SGBD se realizează prin comenzi specifice limbajului SQL. Nivelele conceptual şi intern nefiind distinct delimitate sunt adresate printr-un limbaj comun numit DDL – Data Definition Language, utilizat pentru administrarea şi proiectarea bazei de date în definirea ambelor scheme. Dacă SGBD are o delimitare clară între nivelul conceptual şi cel intern, atunci DDL se foloseşte pentru comenzile la nivel conceptual, iar pentru specificarea schemei interne se foloseşte limbajul SDL – Storage Definition Language. Pentru cel de al treilea nivel, extern, se foloseşte limbajul VDL – View Definition Language, destinat utilizatorilor şi pentru interfața acestora cu nivelul conceptual. Pentru operațiile tipice legate de căutare, inserare, ştergere şi modificarea datelor, SGBD dispune de un limbaj de manipulare numit DML - Data Manipulation Language.

1.5.1. Interfețe SGBD

Un SGBD este un ansamblu complex de programe care asigură interfața între o bază de date şi utilizatorii acesteia. SGBD este componenta software a unui sistem de baze de date care interacționează cu toate celelalte componente ale acestuia asigurând legătura şi interdependența între ele.

Un SGBD oferă interfețele corespunzătoare tuturor categoriilor de utilizatori pentru a facilita legătura acestora cu baza de date. Principalele tipuri de interfețe:

- *Interfețe pe bază de meniuri* – care oferă utilizatorilor o listă de opțiuni (meniuri) pentru formularea interogărilor.

- *Interfețe grafice* – afişează utilizatorului un set de diagrame, cererile sunt formulate prin manipularea acestor diagrame. De cele mai multe ori interfețele grafice sunt asociate cu meniurile.
- *Interfețe bazate pe videoformate* – se utilizează pentru introducerea de noi date, actualizarea bazei de date şi căutare.
- *Interfețe în limbaj natural* – acceptă comenzi scrise în limba engleză sau alte limbi de circulație internațională. Interpretarea cererilor se face pe baza unui set de standard de cuvinte cheie ce sunt interpretate pe baza schemei interne.
- *Interfețe specializate pentru cereri repetate* (limbaj de comandă) – sunt destinate unei anumite categorii de utilizatori, de exemplu pentru angajații unei bănci se implementează un mic set de comenzi prescurtate pentru a micşora timpul necesar introducerii comenzii.
- *Interfețe pentru administrarea bazei de date* – se utilizează pentru comenzile privilegiate utilizate de administratorii bazei de date şi se referă la crearea de conturi, parole, setarea parametrilor sistemului, autorizarea intrării pe un anumit cont, reorganizarea structurii de stocare a datelor din baza de date, accsul la e, înregistrări.

Exemple de SGBD: MySQL, Microsoft SQL, Microsoft Access, Visual FoxPro, Oracle.

1.5.2. Funcții şi servicii oferite de un SGBD

Un SGBD trebuie să asigure funcțiile (Figura 4):

- *funcția de descriere a datelor* – se face cu ajutorul LDD, realizându-se descrierea atributelor din cadrul structurii bazei de date, legăturile dintre entitățile bazei de date, se definesc eventualele criterii de validare a datelor (§5.1.3.), metode de acces la date, integritatea datelor. Concretizarea acestei funcții este schema bazei de date.
- *funcția de manipulare* – este cea mai complexă şi realizează actualizarea şi regăsirea datelor.
- *funcția de utilizare* – asigură mulțimea interfețelor necesare pentru comunicare a tuturor utilizatorilor cu baza de date.

Categorii de utilizatori:

neinformaticieni – beneficiarii informației, nu trebuie să cunoască structura bazei de date, nu trebuie să programeze aplicații, ci doar să le folosească prin intermediul unei interfețe suficient de prietenoase.

informaticieni – crează structura bazei de date şi realizează procedurile complexe de exploatare a bazei de date;

administratorul bazei de date – utilizator special, cu rol hotărâtor în funcționarea optimă a întregului sistem.

- *funcția de administrare* – administratorul este cel care realizează schema conceptuală a bazei de date, iar în perioada de exploatare a bazei de date autorizează accesul la date, reface baza în caz de incident.
- *funcția de protecție a bazei de date* – ansamblul de măsuri necesare pentru asigurarea integrității (semantică, acces concurent, salvare/restaurare) şi securității datelor (autorizare acces, utilizare viziuni, criptare).

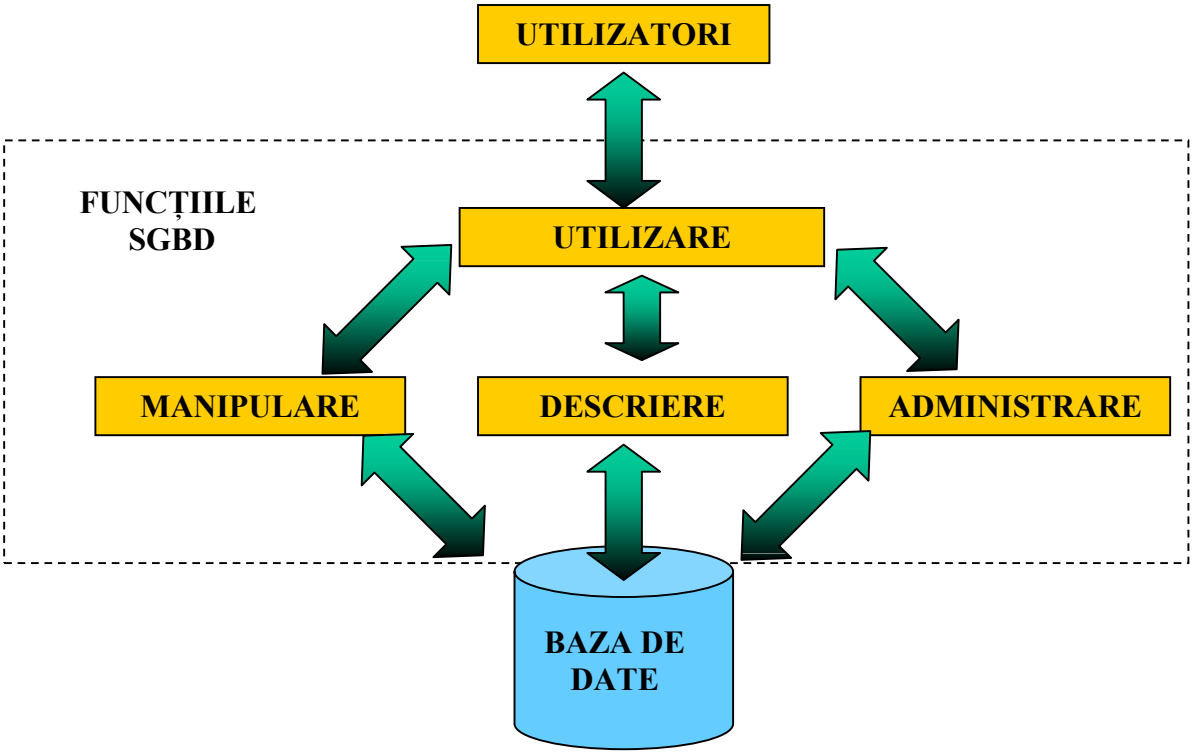


Fig. 4

Prin acestea, un SGBD trebuie să asigure:

- *definirea – crearea bazei de date*;
- *introducerea* (*adăugarea*) *datelor* în baza de date;
- *modificarea* structurii sau a unor date deja existente în baza de date;
- *ştergerea* datelor din baza de date;
- *consultarea bazei de date* – interogare/extragerea datelor.

În plus un SGBD mai asigură şi alte *servicii*:

- *suport pentru limbaj de programare*;
- *interfață* cât mai atractivă pentru comunicarea cu utilizatorul;
- *tehnici avansate de memorare, organizare, accesare* a datelor din baza de date;
- *utilitare* încorporate: sistem de gestiune a fişierelor, listelor, tabelelor etc.;
- "*help*" pentru ajutarea utilizatorului în lucrul cu baza de date.

Apariția şi răspândirea rețelelor de calculatoare a dus la dezvoltarea SGBD-urilor în direcția *multiuser*: mai mulți utilizatori folosesc simultan aceeaşi bază de date.

Principalul avantaj al rețelelor a fost eficiența mult sporită de utilizare a resurselor sistemelor de calcul: la o bază de date aflată pe un server central au acces simultan mai mulți utilizatori, situați la distanță de server, de unde rezultă o bună utilizare a resurselor server-ului şi o economie de memorie datorată memorării unice a bazei de date.

Un SGBD este dotat cu un *limbaj neprocedural de interogare a bazei de date – SQL –* care permite accesul rapid şi comod la datele stocate în baza de date.

1.5.3. Activitățile asigurate de SGBD

Un SGBD trebuie să asigure următoarele activități:

- *definirea şi descrierea structurii bazei de date* – se realizează printr-un limbaj propriu, limbaj de definire a datelor (LDD), conform unui anumit model de date;
- *încărcarea datelor în baza de date* – se realizează prin comenzi în limbaj propriu, limbaj de manipulare a datelor (LMD);
- *accesul la date* – se realizează prin comenzi specifice din limbajul de manipulare a datelor. Accesul la date se referă la operațiile de *interogare şi actualizare*.

Interogarea este complexă şi presupune vizualizarea, consultarea, editarea de situații de ieşire (rapoarte, liste, regăsiri punctuale).

Actualizarea presupune trei operațiuni: *adăugare*, *modificare* (efectuate prin respectarea restricțiilor de integritate ale bazei de date) şi *ştergere*;

- *întreținerea bazei de date* – se realizează prin utilitare proprii ale SGBD;
- *reorganizarea bazei de date* – se face prin facilități privind actualizarea structurii bazei de date şi modificarea strategiei de acces. Se execută de către administratorul bazei de date;

- *securitatea datelor* – se referă la asigurarea *confidențialității* datelor prin autorizarea şi *controlul accesului* la date, *criptarea datelor, realiarea de copii ale programelor şi fişierelor de bază*.

1.5.4. Obiectivele unui SGBD

Un SGBD are rolul de a furniza suportul software complet pentru dezvoltarea de aplicații informatice cu baze de date.

El trebuie să asigure:

- *minimizarea costului de prelucrare a datelor,*
- *reducerea timpului de răspuns*,
- *flexibilitatea aplicațiilor* şi
- *protecția datelor*.

Pentru satisfacerea performanțelor enumerate, SGBD trebuie să asigure un minim de *obiective*.

1. *Asigurarea independenței datelor* – trebuie privită din două puncte de vedere:
   - *independența logică* – se referă la posibilitatea adăgării de noi tipuri de înregistrări de date sau extinderea structurii conceptuale, fără a determina rescrierea programelor de aplicație;
   - *independența fizică* – modificarea tehnicilor fizice de memorare fără a determina rescrierea programelor de aplicație.
2. *Asigurarea redundanței minime şi controlate a datelor* –stocarea informațiilor în bazele de date se face astfel încât datele să nu fie multiplicate. Totuşi, pentru a îmbunătăți performanțele legate de timpul de răspuns, se acceptă o anumită redundanță a datelor, controlată, pentru a asigura coerența bazei de date şi eficiența utilizării resurselor hardware.
3. *Asigurarea facilităților de utilizare a datelor* – presupune ca SGBD-ul să aibă anumite componente specializate pentru:

- *folosirea datelor de către mai mulți utilizatori în diferite aplicații* – datele de la o aplicație trebuie să poată fi utilizate şi în alte aplicații.
- *accesul cât mai simplu al utilizatorilor la date* – fără ca ei să fie nevoiți să cunoască structura întregii baze de date; această sarcină cade în seama administratorului bazei de date.
- *existența unor limbaje performante de regăsire a datelor* – care permit exprimarea interactivă a unor cereri de regăsire a datelor.

- *sistemul de gestiune trebuie să ofere posibilitatea unui acces multicriterial la informațiile din baza de date* – spre deosebire de sistemul clasic de prelucrare pe fişiere unde există un singur criteriu de adresare, cel care a stat la baza organizării fişierului.

4. *Asigurarea securității datelor împotriva accesului neautorizat*.
5. *Asigurarea coerenței şi integrității datelor împotriva unor ştergeri intenționate sau neintenționate* – se realizează prin intermediul unor proceduri de validare, a unor protocoale de control concurent şi a unor proceduri de refacere a bazei de date.
6. *Asigurarea partajabilității datelor* – se referă pe de o parte la asigurarea accesului mai multor utilizatori la aceleaşi date şi de asemenea la posibilitatea dezvoltării unor aplicații fără a se modifica structura bazei de date.
7. *Asigurarea legăturilor între date* – corespund asocierilor care se pot realiza între obiectele unei aplicații informatice. Orice SGBD trebuie să permită definirea şi descrierea structurii de date, precum şi a legăturilor dintre acestea, conform unui model de date (de exemplu, modelul relațional).
8. *Administrarea şi controlul datelor* – sunt asigurate de SGBD, în sensul că datele pot fi folosite de mai mulți utilizatori în acelaşi timp, iar utilizatorii pot avea cerințe diferite şi care pot fi incompatibile. SGBD trebuie să rezolve probleme legate de concurență la date, problemă care apare mai ales în lucrul în mediu de rețea de calculatoare.

2. Etapele de realizare a unei bănci de date

2.1. Etapa de analiză

Proiectarea unei baze de date constă din proiectare logică şi fizică a acesteia astfel încât să răspundă cerinţelor utilizatorilor pentru un anumit set de aplicaţii specifice. În general, proiectarea corectă a unei baze de date parcurge etapele de analiză şi programare. Pentru etapa de analiză se recomandă parcurgerea paşilor:

1. *Studierea problemei de rezolvat* – constă din studiul şi descrierea activităţilor pentru care se va organiza baza de date (cerinţe şi resurse);
2. *Proiectarea structurii bazei de date* – are ca principale activităţi alegerea modelului de SGBD, proiectarea funcţiilor bazei de date, stabilirea intrărilor şi ieşirilor aplicaţiilor;
3. *Stabilirea modului de memorare a datelor* – în memorie (temporare) sau pe disc în baza de date, pentru care trebuie stabilită şi structura;
4. *Stabilirea algoritmului general de rezolvare a problemei* – la nivel de schemă bloc;
5. *Stabilirea structurii meniului principal al aplicaţiei* – acesta trebuie să conţină opţiuni pentru toate funcţiunile sistemului informatic care sunt accesibile utilizatorului.
6. *Împărţirea aplicaţiei pe programe* – fiecare program urmăreşte rezolvarea unei părţi a problemei generale.

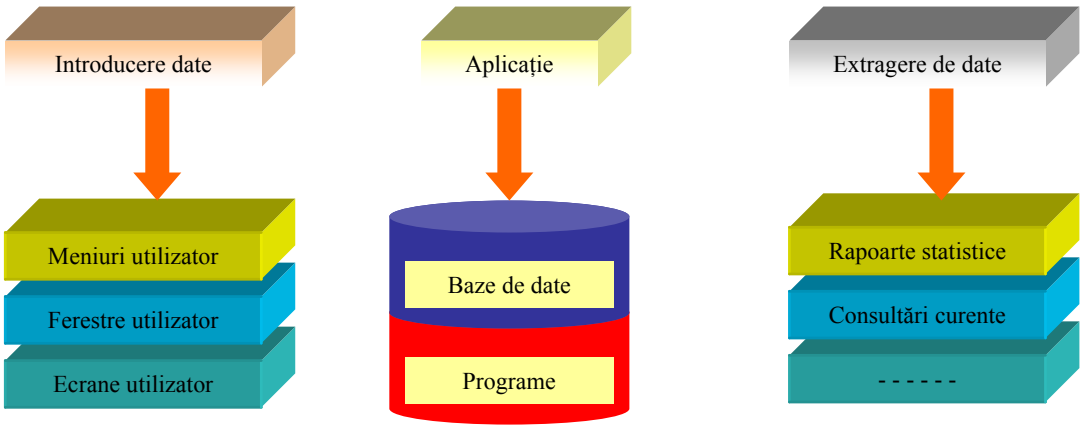
2.2. Etapa de programare

În etapa de programare se recomandă urmărirea paşilor:

1. *Elaborarea programelor:* realizarea componentelor logice – programe/aplicaţii – începând cu programul monitor (principal), continuând cu cele de introducere/actualizare a datelor, cu cele de prelucrare şi terminând cu programele de interogare baza de date şi extragere de informaţii.
2. *Testarea funcţionării corecte a programelor individuale şi a sistemului în ansamblu* – pentru cât mai multe situaţii posibile (ideal ar fi testarea în toate situaţiile posibile, dar numărul acestora este foarte mare) incluzând în mod obligatoriu şi situaţiile limită.
3. *Înlăturarea erorilor depistate* – şi reluarea pasului 2. Dacă nu mai sunt erori, se continuă cu următoarea etapă.

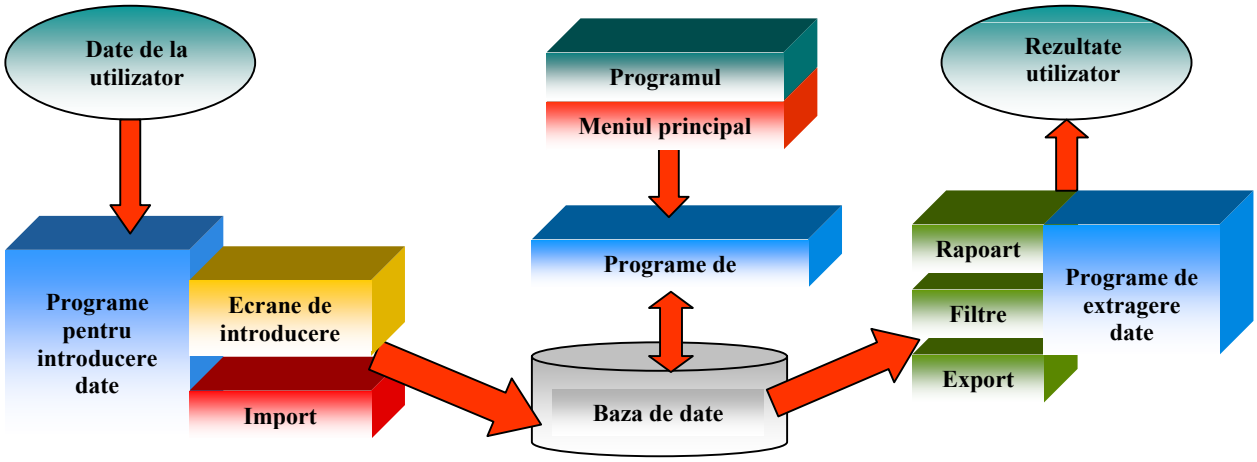
2.3. Punerea în funcțiune şi exploatarea bazei de date

1. *Încărcarea şi manipularea datelor* – operații de actualizare şi consultare, întreținere (dezvoltare) a bazei de date.
2. *Elaborarea documentației de utilizare* – care se predă beneficiarului, cât şi documentația tehnică a sistemului (de preferat ca fiecare program să aibă câte o fişă tehnică, pentru o mai uşoară depanare, întreținere).



Schema bloc a unei aplicații din punct de vedere utilizator

Fig. 5



Schema bloc a unei aplicații din punct de vedere al proiectantului

Fig. 6

Notă: Programele de extragere date pot fi elaborate de programator sau pot fi obținute cu generatoarele incluse în SGBD.

Întreaga activitate din sistem este controlată de programul monitor care conține un meniu principal definit de utilizator. Programele aplicației pentru introducerea datelor, prelucrare, actualizare şi extragere de date sub formă de rapoarte se află sub controlul programului monitor.

2.4. Documentația aplicației

Deoarece etapa de analiză este proprie fiecărei aplicații în parte, nu se recomandă detalierea etapelor 1 ÷6. De aceea în cele ce urmează este detaliată documentarea programelor prin fişa tehnică, fişă care trebuie să conțină rubricile:

Program – nume;

Scop;

Parametri;

Variabile – de intrare, de ieşire;

Ce programe apelează;

Care sunt programele care îl apelează;

Ce fişiere utilizează;

Explicații - sub formă de comentarii în programele sursă;

Programul sursă.

În §12.2.2 se arată cum se obține în mod automat, cu ajutorul VFP, documentarea unei aplicații realizată cu acest sistem.

3. Sisteme de gestiune a bazelor de date relaționale

3.1. Modelul relațional al datelor

Un sistem de gestiune a bazelor de date relaționale se defineşte ca fiind un sistem de gestiune care utilizează organizarea datelor conform modelului relațional. Conceptul de bază al modelului relațional este acela de relație/tabelă (limbajul SQL specializat în comenzi de manipulare la nivel de tabelă).

3.1.1. Structura relațională a datelor

Structura relațională a datelor cuprinde următoarele componente.

Domeniul – este ansamblul de valori caracterizat printr-un nume (domeniu de valori). El poate fi precizat explicit – prin enumerarea tuturor valorilor care aparțin domeniului – sau implicit prin precizarea proprietăților pe care le au valorile din domeniu. Pentru un ansamblu de valori $D_1,...,D_n$ produsul cartezian al acestora reprezintă ansamblul tuplurilor $(v_1,...,v_n)$, unde v_1 este o valoare din D_1, v_2 este o valoare din D_2, etc., iar tuplura corespunde unei linii din tabelă.

Relația (tabela) – este un subansamblu al produsului cartezian al mai multor domenii, caracterizat prin nume şi care conține tupluri cu semnificație (tabelă). Într-o relație, tuplurile trebuie să fie distincte – nu se admit duplicate. O reprezentare a relației este tabelul bidimensional (tabela de date) în care liniile reprezintă tuplurile, iar coloanele corespund domeniilor. Numărul tuplurilor dintr-o tabelă este *cardinalul tabelei*, numărul valorilor dintr-un tuplu este *gradul tabelei*. Pentru a diferenția coloanele care conțin valori ale aceluiaşi domeniu, eliminând dependența de poziție, se asociază fiecărei coloane un nume distinct – *atribut*. În timp ce tuplurile sunt unice, un domeniu poate apărea de mai multe ori în produsul cartezian pe baza căruia este definită tabela.

Tabelul 1 prezintă comparativ conceptele utilizate în organizarea datelor în fişiere, în baze de date, conceptele din teoria relațională şi din SGBDR.

Tabelul 1

Fişiere	Teoria BD	Teoria relaţională	SGBDR
Fişier	Colecţie de date	Relaţie	Tabelă
Înregistrare	Familie de caracteristici	Tuplu	Linie
Câmp Valoare	Caracteristică Domeniu de Valori	Atribut Domenii	Nume coloană Domeniu coloană

3.1.2. Operaţii în algebra relaţională

Reuniunea – operaţie în algebra relaţională, definită pe două relaţii R_1 şi R_2, ambele cu aceeaşi schemă (structură), ce constă din construirea unei noi relaţii R_3, cu schema identică cu R_1 şi R_2, având drept extensie (înregistrările) tuplurile din R_1 şi R_2 luate împreună, o singură dată.

Diferenţa – definită pe două relaţii R_1 şi R_2, cu aceeaşi schemă, având drept extensie tupluri ale relaţiei R_1 care nu se regăsesc în relaţia R_2.

Produs cartezian – definit pe două relaţii R_1 şi R_2, operaţie care constă din construirea unei noi relaţii R_3, a cărei schemă se obţine din concatenarea schemelor relaţiilor R_1 şi R_2, şi a cărei extensie cuprinde toate combinaţiile tuplurilor din R_1 cu cele din R_2.

Proiecţia – operaţie din algebra relaţională definită asupra unei relaţii R, ce constă din construirea unei noi relaţii P, în care se regăsesc numai acele atribute din R specificate explicit în cadrul operaţiei. Suprimarea unor atribute din R însemnă efectuarea unor tăieturi verticale asupra lui R şi pot avea ca efect apariţia unor tupluri duplicate, care se cer a fi eliminate. În practică, operaţia de proiecţie se face considerând şi un câmp care conţine o cheie unică, astfel încât fiecare înregistrare devine unică.

Selecţia – operaţie în algebra relaţională definită asupra unei relaţii R, care constă din construirea unei relaţii S, a cărei schemă este identică cu cea a relaţiei R şi a cărei extensie este constituită din acele tupluri din R care satisfac o condiţie explicită în cadrul relaţiei. Deoarece nu toate tuplurile din R satisfac condiţia, selecţia înseamnă efectuarea de tăieturi pe orizontală asupra relaţiei R, adică eliminarea de tupluri. Condiţia precizată în cadrul operaţiei de selecţie este de forma:

<Atribut> <operator de compataţie> <Valoare>

Joncțiunea (join-ul) – operație în algebra relațională definită pe două relații R_1 şi R_2, care constă din construirea unei noi relații R_3, prin concatenarea unor tupluri din R_1 cu tupluri din R_2. Se concatenează acele tupluri din R_1 şi R_2 care satisfac o anumită condiție, specificată explicit în cadrul operației. Extensia relației R_3 va conține combinațiile acelor tupluri care satisfac condiția de concatenare. Cel mai important tip de join din punct de vedere al utilizării este *equijoin*-ul, care este o joncțiune dirijată de o condiție de forma:

<Atribut din R_1> = <Atribut din R_2>

Joncțiunea naturală se referă la equijoin-ul aplicat tuturor câmpurilor comune. Operatorul pentru joncțiunea naturală este * .

Joncțiunea exterioară (*outer join*) include trei tipuri de joncțiune:

- *Joncțiunea exterioară stânga* ⟕ care include toate tuplurile din partea stângă a relației, iar în partea dreaptă a relației include numai acele tupluri care au corespondent.
- *Joncțiunea exterioară dreapta* ⟖ care include toate tuplurile din partea dreaptă a relației, iar în partea stângă a relației include numai acele tupluri care au corespondent.
- *Joncțiunea exterioară completă* ⟗ care include toate tuplurile din stânga şi din dreapta relației.

Intersecția – operație din algebra relațională definită pe două relații R_1 şi R_2, ambele cu aceeaşi schemă, care constă din construirea unei noi relații R_3, cu schema identică cu a operanzilor şi cu extensia formată din tuplurile din R_1 şi R_2.

Diviziunea – operație din algebra relațională definită asupra unei relații R, care constă din construirea cu ajutorul unei relații r a relației Q. Tuplurile relației Q concatenate cu tuplurile relației r permit obținerea tuplurilor relației R.

Exemple

Pentru a înțelege mai bine aceste operații să presupunem că avem următoarele trei structuri de date:

1. *Y*1 – fişier închirieri maşini

Sofer_id	Auto_id	Ziua
22	101	10/10/2004
58	103	11/12/2004

2. *S*2 – fişier clienţi

Sofer_id	Sofer _nume	Permis_ani
22	Ionescu Paul	7
31	Leahu Mihai	8
58	Arsene Ion	10

3. *S*3 – fişier clienţi

Sofer_id	Sofer _nume	Permis_ani
28	Doncea Horia	5
31	Leahu Mihai	8
44	Coman Stefan	12
58	Arsene Ion	10

Semnificaţia structurilor de date este următoarea:

- în relaţia *Y*1 se stochează informaţii referitoare la maşinile rezervate cu atributele: *Sofer_id* – codul şoferului, *Auto_id* – codul maşinii închiriate, *Ziua* – data la care a fost închiriată;
- în relaţiile *S*2 şi *S*3 se stochează informaţii referitoare la şoferi (fragmentat), cu următoarele atribute: *Sofer_id* – codul şoferului (pentru a stabili o legătură între structura de date *Y*1 şi *S*2, *S*3), *Sofer_nume* – numele şoferului, *Permis_ani* – vechimea permisului de conducere.

Principalele operaţii:

- *Reuniunea* $R3 = S2 \cup S3$

Sofer_id	Sofer _nume	Permis_ani
22	Ionescu Paul	7
31	Leahu Mihai	8
58	Arsene Ion	10
44	Coman Stefan	12
28	Doncea Horia	5

- *Intersecția* $R3 = S2 \cap S3$

Sofer_id	Sofer _nume	Permis_ani
31	Leahu Mihai	8
58	Arsene Ion	10

- *Diferența* $S2 - S3$

Sofer_id	Sofer _nume	Permis_ani
22	Ionescu Paul	7

- *Produs cartezian,* $Y1 \times S2$. Fiecare linie din relația $Y1$ se combină cu fiecare linie din relația $S2$; ca rezultat al produsului cartezian, apare un conflict deoarece ambele relații $Y1$ şi $S2$ au câte un câmp cu numele *sofer_id*, care poate fi rezolvat prin operația de *redenumire*.

Query

Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_ani
22	101	10.10.04	22	Ionescu Paul	7
22	101	10.10.04	31	Leahu Mihai	8
22	101	10.10.04	58	Arsene Ion	10
58	103	11.12.04	22	Ionescu Paul	7
58	103	11.12.04	31	Leahu Mihai	8
58	103	11.12.04	58	Arsene Ion	10

Comanda SQL pentru pentru produs cartezian $Y1 \times S2$ este dată în Figura 1.

Operatorul redenumire se utilizează sub forma:

$$\rho\,(C(1 \rightarrow \text{Sofer_id1}, 5 \rightarrow \text{Sofer_id2}),\ Y1 \times S2)$$

cu următoare interpretare: se redenumeşte coloana 1 în *Sofer_id*1 şi coloana 5 în *Sofer_id*2, în relația dată de produsul cartezian obținut din relațiile $Y1$ şi $S2$.

- *Proiecția.* Deoarece valorile dintr-un domeniu se pot repeta este necesară eliminarea repetărilor:

$\pi_{Sofer_nume,Permis_ani}(S3)$

Sofer _nume	Permis_ani
Doncea Horia	5
Leahu Mihai	8
Coman Stefan	12
Arsene Ion	10

- *Selecția* $\sigma_{Permis_ani > 8}(S3)$

Sofer_id	Sofer _nume	Permis_ani
44	Coman Stefan	12
58	Arsene Ion	10

Din relația *S*3 au fost selectate numai acele înregistrări care satifac condiția de vechime a permisului (mai mult de 8 ani).

Operația de proiecție se poate combina cu cea de selecție pentru optimizarea cererilor de date; de exemplu dacă dorim extragerea datelor din relația *S*3 numai pentru câmpurile *Sofer_nume* şi *Permis_ani* mai mare de 8 ani:

$$\pi_{Sofer_nume,Permis_ani}(\sigma_{Permis_ani > 8}(S3))$$

Sofer _nume	Permis_ani
Coman Stefan	12
Arsene Ion	10

Joncțiunea (join-ul). Operația de joncțiune condiționată a două relații este echivalentă cu selecția condiționată a produsului cartezian al celor două relații.

$$S2 \bowtie_{S2.Sofer_id < Y1.Sofer_id} Y1$$

(Sofer_id)	Sofer _nume	Permis_ani	(Sofer_id)	Auto_id	Ziua
22	Ionescu Paul	7	58	103	11/12/2004
31	Leahu Mihai	8	58	103	11/12/2004

Operația de joncțiune condiționată mai poartă uneori şi numele de *theta-join*.

Pentru acest exemplu, *equijoin*-ul, caz special al joncțiunii condiționate în care condiția *c* conține doar egalitați, devine

$$S2 \bowtie_{Sofer_id} Y1$$

Sofer_id	Sofer _nume	Permis_ani	Auto_id	Ziua
22	Ionescu Paul	7	101	10/10/2004
58	Arsene Ion	10	103	11/12/2004

Noua relație rezultată din equijoin este similară cu produsul cartezian dintre *S*2 şi *Y*1, dar numai o copie a câmpului pentru care este specificată egalitatea, va fi specificată în noua relație (nu mai apare duplicarea câmpului).

Exemple de *joncțiuni exterioare*

- joncțiune exterioară stânga

$$\bowtie_{Y1.Sofer_id=S2.Sofer_id} Y1 \quad S2$$

Test

Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_ani
22	101	10.10.04	22	Ionescu Paul	7
58	103	11.12.04	58	Arsene Ion	10

- joncțiune exterioară dreapta

$$\bowtie_{Y1.Sofer_id=S2.Sofer_id} Y1 \quad S2$$

Test1

Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_ani
22	101	10.10.04	22	Ionescu Paul	7
.NULL.	.NULL.	.NULL.	31	Leahu Mihai	8
58	103	11.12.04	58	Arsene Ion	10

- joncțiune exterioară competă

$$\bowtie_{Y1.Sofer_id=S2.Sofer_id} Y1 \quad S2$$

Test2

Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_ani
22	101	10.10.04	22	Ionescu Paul	7
58	103	11.12.04	58	Arsene Ion	10
.NULL.	.NULL.	.NULL.	31	Leahu Mihai	8

De remarcat prezența câmpurilor cu valoarea **NULL** utilizată în joncțiuni pentru a specifica lipsa de date.

Presupunând că tabele au fost create, programul sursă pentru testarea celor trei tipuri de joncțiuni exterioare este prezentat în Figura 1.

test_sql.prg

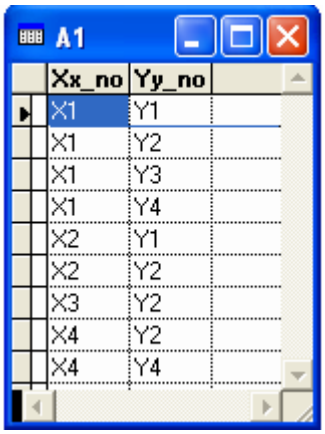
```
close database
* Operatii JOIN
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_ani from ;
y1 a LEFT OUTER JOIN s2 b ;
on a.sofer_id=b.sofer_id into table test
browse
close database
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_ani from ;
y1 a RIGHT OUTER JOIN s2 b on a.sofer_id=b.sofer_id into table test1
browse
close database
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_ani from ;
y1 a FULL OUTER JOIN s2 b on a.sofer_id=b.sofer_id into table test2
browse
* Produs cartezian
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_ani ;
from y1 a,s2 b
* Diviziune
select distinct xx_no from A1 where not exists (select yy_no from B1 where ;
A1.yy_no=B1.yy_no)
select distinct xx_no from A1 where not exists (select yy_no from B2 where ;
A1.yy_no=B2.yy_no)
select distinct xx_no from A1 where not exists (select yy_no from B3 where ;
A1.yy_no=B3.yy_no)
close database
```

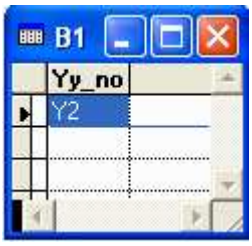
Fig. 1

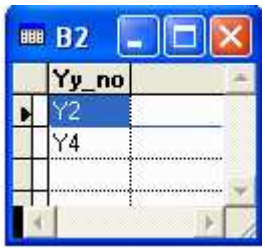
Diviziunea –Se utilizează la exprimarea interogărilor de tipul: "*Găseşte automobilistul care a rezervat toate automobilele*".

$$A / B = \{\langle x \rangle \mid \neg\exists \langle x, y \rangle \in A,\ \forall \langle y \rangle \in B\}$$

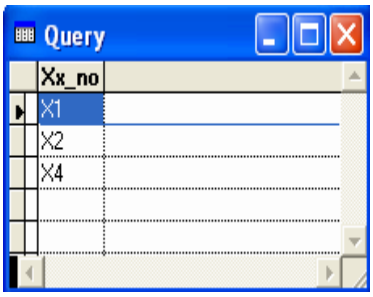
relația obținută prin diviziune va conține toate tuplurile *x* care pentru fiecare tuplură *y* din *B* nu are asociată un tuplu *xy* în *A*. Pentru a înțelege mai bine operația de diviziune vom considera următorul exemplu, în care considerăm tabelul *A*1 (câmpuri ***xx_no, yy_no***) şi tablele *B*1, *B*2, *B*3 (un singur câmp ***yy_no,*** corespondent cu cel din tabelul *A*) şi realizăm operațiile de diviziune *A*1/*B*1, *A*1/*B*2, *A*1/*B*3:



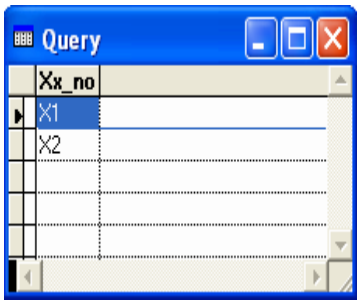




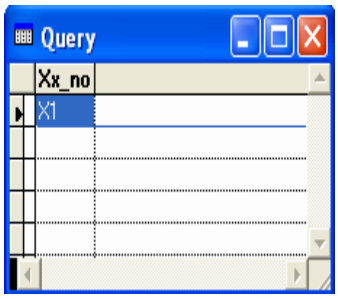
Secvența de program care realizează cele trei operații de diviziue se găseşte în figura 1, ultima secțiune **Diviziune*. Rezultatul celor trei comenzi SQL este:



*A*1/*B*1



*A*1/*B*2



*A*1/*B*3

3.1.3. Optimizarea cererilor de date

Se realizează în două etape:

1. exprimarea cererilor de date sub forma unor expresii algebrice relaționale care au la bază echivalența dintre calculul relațional şi algebra relațională,

2. aplicarea unor transformări algebrice asupra expresiilor obținute în etapa precedentă, în scopul obținerii unor expresii echivalente cu cele inițiale, dar care să fie executate mai eficient.

Proprietăți

Comutativitatea operațiilor de join şi produs cartezian:

$E_1 \bowtie E_2 = E_2 \bowtie E_1$

$E_1 \times E_2 = E_2 \times E_1$

Asociativitatea operațiilor de join şi produs cartezian:

$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

$(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$

Compunerea proiecțiilor

Compunerea selecțiilor $\sigma_{F_1}(\sigma_{F_2}(E)) = \sigma_{F_1 \cap F_2}(E)$, deoarece $F_1 \bigcap F_2 = F_2 \bigcap F_1$, selecțiile se pot comuta:

$$\sigma_{F1}(\sigma_{F2}(E)) = \sigma_{F2}(\sigma_{F1}(E))$$

Comutarea selecției cu proiecția

Comutarea selecției cu produsul cartezian

Comutarea selecției cu reuniunea

Comutarea selecției cu diferența

Comutarea proiecției cu produsul cartezian

Comutarea proiecției cu reuniunea

Prin deplasarea operațiilor de selecție cât mai la stânga expresiilor algebrice se reduce numărul de tupluri care trebuie manipulate în procesul de executare a cererii.

Se pot menționa următoarele strategii de optimizare a cererilor de date:

- Deplasarea operației de selecție înaintea operației de joncțiune – joncțiunea şi produsul cartezian acționează ca generatori de tupluri. Prin selecție se reduce dimensiunea relațiilor la care se aplică aceşti generatori de tupluri. Se ține seama că operația de joncțiune poate fi exprimată sub forma unui produs cartezian urmat de o selecție, iar în cazul joncțiunii naturale printr-un produs cartezian urmat de de o selecție şi de o proiecție.
- Deplasarea operațiilor de proiecție înaintea operațiilor de joncțiune – se realizează prin folosirea proprietății de comutare a selecției cu produsul cartezian
- Combinarea selecțiilor multiple – se realizează cu ajutorul relației de compunere a selecțiilor.

- Deplasarea operațiilor de selecție înaintea operațiilor de proiecție- realizată pe baza proprietății de comutare a selecției cu proiecția. Eliminarea tuplurilor duplicate obținute prin proiecție se face prin ordonarea tuplurilor. Selecția reduce numărul tuplurilor ce trebuiesc ordonate facilitând operația de proiecție.

3.1.4. Baze de date relaționale

Conceptul bazelor de date relaționale este axat pe metodologia entitate-tabelă (*E-T*).

Entitățile modelează obiectele care sunt implicate într-o organizație, de exemplu studenții, profesorii şi cursurile dintr-o universitate.

Tabelele modelează legăturile dintre entități, de exemplu profesorii predau cursuri. În plus, restricțiile de integritate aplicate entităților şi relațiilor formează o parte importantă a specificațiilor *E-T*, de exemplu, un profesor poate preda un singur curs la o anumită oră dintr-o anumită zi.

Entitățile similare pot fi agregate în tipuri de entități. De exemplu, Ionescu, Popescu, Albu, pot fi agregați în tipul de entitate PERSOANĂ, pe baza faptului că aceste entități sunt oameni. Ionescu şi Albu pot aparține tipului de entitate STUDENT, deoarece aceste obiecte sunt studenți.

La fel ca tabelele, entitățile sunt descrise utilizînd atribute. Fiecare atribut specifică o particularitate semnificativă a entității. De exemplu, atributul *Nume* al unei entități de tip PERSOANĂ specifică şirul de caractere care alcătuieşte numele persoanei din lumea reală. De asemenea, *Vârsta* este atribuitul care specifică de câte ori Pământul a înconjurat Soarele față de momentul în care o anumită persoană s-a născut. Pentru fiecare atribut, se asociază un domeniu care specifică setul de valori pe care le poate lua atributul. În principiu, este posibil ca două entități diferite ale aceluiaşi tip să posede valori identice pentru toate atributele.

Atributelor li se asociază valori care au drept scop identificarea entității, realizându-se o înregistrare în tabela respectivă. Atribute pot fi:

- *complexe* - cele care pot fi divizate în mai multe părți cu semnificație independentă. De exemplu, atributul *Adresa* poate fi divizat în mai multe atribute: *Oraş*, *Cod poştal*, *Stradă*, *Număr*, *Bloc*, *Etaj*, *Nr_Apartament*. Una din cerințele importante referitoare la valorile din domeniu se referă la *atomicitatea datelor*. Atomicitatea datelor nu înseamnă că aceste valori nu se pot descompune, datele pot fi şiruri de caractere, ceea ce înseamnă că pot fi descompuse. Cerința de atomicitate se referă la faptul că modelul relațional nu specifică nici un mijloc pentru a privi în interiorul structurii valorilor, astfel că valorile sunt invizibile pentru operatorii relaționali;

- *cu o singură valoare* – din setul de valori ale atributului pentru fiecare entitate există numai o singură valoare, nerepetabilă la o altă entitate. De exemplu, *CNP* (Codul Numeric Personal) pentru o persoană este unic şi nici o altă persoană nu mai are atribuită aceeaşi valoare a *CNP*;

- *cu set de valori* – atributul poate lua orice valoare din set şi care se poate repeta pentru o altă entitate. Este cazul culorilor (două mărci de maşină diferite pot avea aceaşi culoare) sau gradul cadrelor didactice;

- *derivate* – sunt cele ce se pot determină din alte atribute, de exemplu vârsta unei persoane se poate determina scăzând din data curentă data naşterii persoanei respective, sau determinarea valorii unei entităţi prin produsul dintre preţul unitar şi cantitate. Prezenţa atributelor derivate împreună cu cele primare cresc gradul de redundanţă al datelor din baza de date şi trebuie evitată;

- *fără valoare* (**NULL**) – bazele de date acceptă şi posibilitatea lipsei datelor pentru câmpurile anumitor înregistrări, de exemplu, din atributul *Adresă*, poate lipsi *Numele_blocului* sau *Nr_Scara*.

Cheile de restricţie introduse în modelul relaţional sunt asociate cu tipurile de entităţi, de exemplu, două persoane diferite nu pot avea aceleaşi valori pentru atributele *Nume* şi *Adresa*.

Schema în modelul relaţional se defineşte ca fiind un tip de entitate compus din numele tipului de entitate, colecţia de atribute (cu domeniile asociate şi indicaţii în cazul în care un atribut este complex sau cu valori unice) şi cheile de restricţie.

Reprezentarea diagramelor E-T – tipurile de entităţi reprezentate în diagramele *E-T* sunt sub formă de dreptunghiuri şi atributele sub formă de elipse. Atributele tip set de de valori (modelul relaţional permite atributelor să posede un set de valori din domeniu) sunt reprezentate în elipse duble. Sublinierea unui atribut în diagramă îl desemnează ca fiind cheie primară, adică un atribut cu valoare unică. Figura 2 descrie o diagramă *E-R* pentru entitatea PERSOANA:

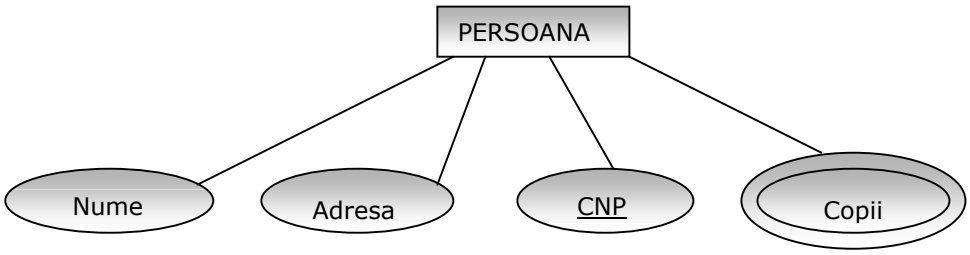


Fig. 2

3.1.5. Tipuri de relaţii

Relația unu-la-unu (1-1) este tipul cel mai simplu de relație, prin care unui element din tabela *R*1 îi corespunde un singur element din tabela *R*2 şi reciproc. Relația 1-1 este mai rar folosită, în general pentru a reduce numărul de atribute dintr-o tabelă, de exemplu dacă vrem să despărțim elementele fixe care se modifică mai rar (datele personale) față de cele care se modifică mai des (situația şcolară anuală).

Relația unu-la-mai mulți (1-*N*) este tipul de relație prin care unui element din tabela *R*1 îi corespund unul sau mai multe elemente din tabela *R*2, iar unui element din tabela *R*2 îi corespunde un singur element din tabela *R*1. De exemplu, presupunem că avem:

- tabela FACULTATE cu atributele *CodFac*, *Denumire*, *Adresa*, *NumeDecan*;
- tabela STUDENTI cu atributele *CodStud*, *CodFac*, *An*, *Grupa*, *Medie*, *Bursa*;

Între cele două tabele există o relație 1-*N* prin intermediul câmpului *CodFac*.

Relația mai mulți-la-mai mulți (*M-N*) prin care unui element din tabela *R*1 îi corespunde unul sau mai multe elemente din tabela *R*2 şi reciproc. De exemplu, un student frecventează mai multe cursuri şi o disciplină este frecventată de mai mulți studenți. Acest tip de relație este frecvent întâlnită, dar nu se poate implementa direct în bazele de date relaționale. Pentru modelare se foloseşte o relație suplimentară, de tip 1-*N* pentru fiecare din tabelele inițiale. Presupunând că avem două tabele MATERII şi STUDENTI se introduce suplimentar tabela NOTE care face legătura între tabelele MATERII şi STUDENTI:

- tabela MATERII are atributele *CodMaterie*, *Denumire*, *An*, *NumeProfesor*;
- tabela STUDENTI cu atributele: *CodStud*, *CodFac*, *An*, *Grupa*, *Medie*, *Bursa*;
- tabela NOTE cu atributele: *CodStud*, *CodMaterie*, *Nota*, *Data*.

Relația unară este acea relație care foloseşte o singură tabelă care este asociată cu ea însăşi. Exemplul clasic este cel al managerului unei companii, care la rândul său este tot un angajat al acestei companii.

Pentru fiecare tip de entitate participantă în cadrul relației se defineşte un rol care primeşte un nume pentru tipul relației. În diagramele *E-T,* tipul relației se reprezintă sub forma unui romb. De exemplu, tipul de relație LUCREAZĂ cu atributul *Data*, definit în Figura 3, are două roluri *Profesor* şi *Departament*:

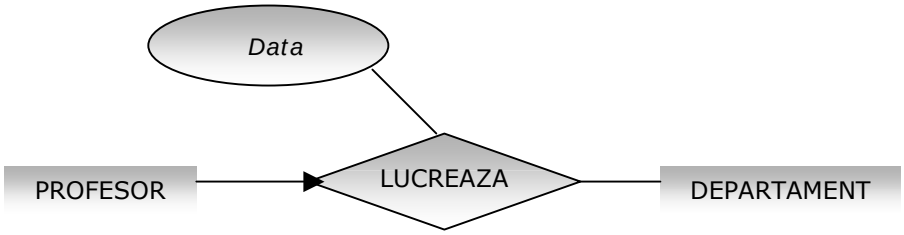


Fig. 3

Rolul *Profesor* identifică entitatea PROFESOR implicată în relaţia LUCREAZĂ, iar rolul *Departament* identifică entitatea DEPARTAMENT în cadrul relaţiei.

În Figura 4 este prezentată o relaţie între trei entităţi: PROIECT (cu rolul *Client*), PARTENER (cu rolul *Produs*), FURNIZOR (cu rolul *Furnizor*), cu tipul de relaţie SOLD şi atributele *Data* şi *Pret*.

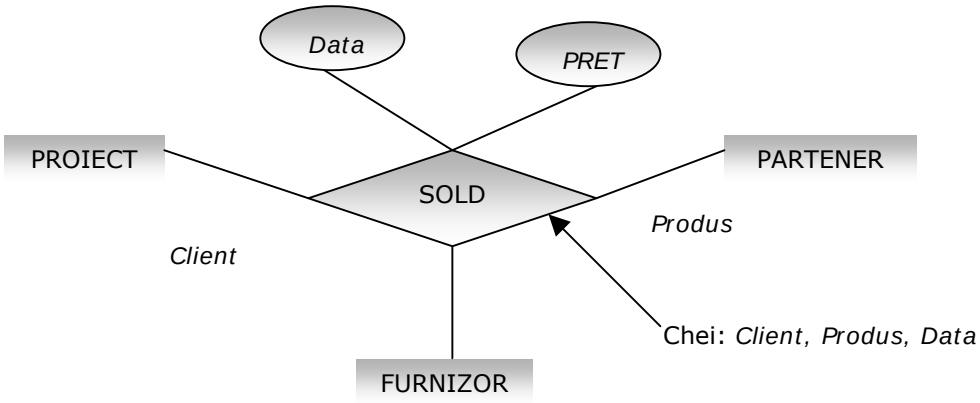


Fig. 4

3.1.6. Restricţia de cardinalitate

Fie *C* un tip de entitate şi *A* tipul de relaţie conectată la *C* prin intermediul rolului *R*. Restricţia de cardinalitate a rolului *R* este o declaraţie de forma *min..max* ataşată la *R*, care restricţionează numărul de instanţe a relaţiilor de tip *A* în care o singură entitate de tip *C* poate participa în rolul *R* şi este un număr în intervalul *min..max*, Figura 5.

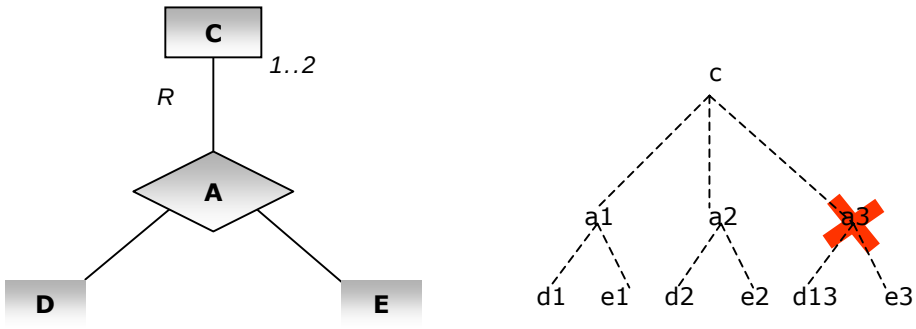


Fig. 5

Figura 5 prezintă şi o diagramă cu restricţia de cardinalitate pentru rolul *R*. În partea dreaptă se prezintă o instanţă validă a diagramei, unde entitatea *C* participă în două relaţii de tip *A*. Relaţia *a*3 este tăiată deoarece violează limitele de cardinalitate 1..2 ale rolului *R*.

În general, modelul *E-T* suportă restricţia de cardinalitate de forma *min..max*, unde *min* este un număr mai mare sau egal cu zero, *max* este un număr mai mare decât zero cu condiţia $min \leq max$. În plus, *max* poate fi simbolizat şi prin caracterul *, care reprezintă infinitul. Astfel, o restricţie de forma 3..* asupra rolului *R* care conectează un tip de entitate *C*, cu un tip de relaţie *A*, înseamnă că fiecare entitate de tip *C* trebuie să participe în rolul *R,* cu cel puţin trei relaţii de tip *A* (fără limită superioară). Pentru restricţiile de cardinalitate de forma *N..N* (unde *min*=*max*) este adesea abreviată cu *N*, iar caracterul * se foloseşte pentru sintagma mai mulţi.

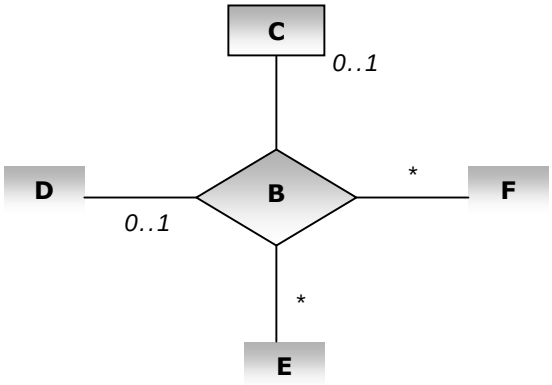


Fig. 6

În Figura 6 sunt prezentate tipuri de relaţii de grad superior (4). Tipul relaţiei este determinat de restricţia de cardinalitate a rolurilor unei relaţii. De exemplu, relaţia de tip *B* implică o corespondenţă unu-la-unu între entitatea *C* şi *D*. Aceasta înseamnă că o entitate de tip *C* poate fi asociată cu cel puţin o entitate de tip *D* şi vice versa. În acelaşi timp, relaţia *B* implică o corespundenţă unu-la-mai mulţi pentru entităţile *E* la *C* şi *D* (ca şi pentru *F* la *C* şi *D*). Aceasta înseamnă că o entitate de tip *E* poate fi asociată cu orice număr (inclusiv zero) de enităţi de tip *C* şi *D*, dar o entitate de tip *C* (sau *D*) poate fi asociată cu cel mult o entitate de tip *E*. Relaţia de corespondenţă unu-la-mai mulţi nu este simetrică; inversa acestei corespondenţe (*C* la *E*) este numită mai *mulţi-la-unul*. Corespondenţa între *E* şi *F* este numită mai *mulţi-la-mai mulţi*, adică o entitate de tipul *E* poate fi asociată cu orice număr de entităţi *F* şi vice versa.

3.2. Regulile lui Codd

Detalierea caracteristicilor, pe care trebuie să le prezinte un SGBD pentru a fi considerat relațional, s-a facut de E. F. Codd în 1985, sub forma a 13 reguli.

R0 – Gestionarea datelor la nivel de relație.
Toate informațiile din baza de date sunt gestionate numai prin mecanisme relaționale. Rezultă că SGBD-ul trebuie să-şi îndeplinească toate funcțiile utilizând ca unitate de informație mulțimea, adică să utilizeze limbaje (SQL) care să opereze la un moment dat pe o întreagă relație.

R1 – Reprezentarea logică a datelor
Informațiile din baza de date relațională trebuie să fie reprezentate explicit la nivel logic într-un singur mod şi anume ca valori în tabelele de date. Rezultă că toate datele trebuie să fie memorate şi prelucrate în acelaşi mod. Informațiile privind numele de tabele, coloane, domenii, definițiile tabelelor virtuale, restricțiile de integritate trebuie să fie memorate tot în tabele de date (catalog).

R2 – Garantarea accesului la date
Accesarea informațiilor din baza de date relațională se va face prin specificarea numelui tabelei, a valorii cheii primare şi numelui de coloană.

R3 - Valorile **NULL**
SGBD trebuie să permită declararea şi manipularea valorilor null, cu semnificația unor date lipsă sau inaplicabile. Valorile **NULL**, care diferă de şirurile de caractere spațiu sau de şirurile vide de caractere sunt importante în implementarea restricțiilor de integritate (integritatea entității şi integritatea referențială) din modelul relațional.

R4 - Metadatele
Informațiile despre descrierea bazei de date, metadatele, trebuie să se prezinte la nivel logic în acelaşi mod cu descrierea datelor propriu-zise, utilizatorul aplicând asupra descrierii bazei de date aceleaşi operații ca şi la datele obişnuite. Sistemul nu trebuie să facă diferențe între descrierea datelor şi a metadatelor utilizând o singură structură, cea relațională.

R5 – Facilitățile limbajelor utilizate
Un sistem relațional trebuie să facă posibilă utilizarea mai multor limbaje, în mai multe moduri. Trebuie să existe cel puțin un limbaj de nivel înalt ale cărui instrucțiuni să poată exprima oricare din

următoarele operații: definirea tabelelor de bază, definirea tabelelor virtuale, manipularea datelor, definirea restricțiilor de integritate, autorizarea accesului, precizarea limitelor tranzacțiilor.

R6 – Actualizarea tabelelor virtuale
Nu toate atributele din cadrul unei tabele virtuale, deci nu toate tabelele virtuale, sunt teoretic actualizabile.

R7 – Actualizările în baza de date
SGBD trebuie să ofere posibilitatea manipulării unei tabele (de bază sau virtuale) nu numai în cadrul operațiilor de regăsire, ci şi în cazul celor de adăugare, modificare şi ştergere a datelor. În cursul operațiilor prin care se schimbă conținutul bazei de date să se lucreze la un moment dat pe o întreagă relație.

R8 – Independența fizică a datelor
Programele de aplicație nu trebuie să fie afectate de schimbările efectuate în modul de reprezentare a datelor sau în metodele de acces. O schimbare a structurii fizice a datelor nu trebuie să blocheze funcționarea programelor de aplicație.

R9 – Independența logică a datelor
Programele de aplicație nu trebuie să fie afectate de schimbările efectuate asupra relațiilor bazei de date.

R10 – Restricțiile de integritate
Restricțiile de integritate trebuie să poată fi definite în limbajul utilizat de SGBD pentru definirea datelor şi să fie memorate

R11 – Distribuirea geografică a datelor
În situația în care datele sunt distribuite, programele de aplicație să fie logic aceleaşi cu cele utilizate în cazul în care datele sunt fizic centralizate. Utilizatorul trebuie să perceapă datele ca fiind centralizate. Localizarea şi recompunerea datelor distribuite cade în seama sistemului.

R12 – Prelucrarea datelor la nivel de bază

Dacă SGBD posedă un limbaj de bază de nivel scăzut orientat pe prelucrarea de înregistrări (tupluri) şi nu pe prelucrarea mulțimilor (relațiilor) acest limbaj nu trebuie folosit, pentru a se evita restricțiile de integritate sau restricțiile introduse prin utilizarea limbajelor relaționale de nivel înalt

3.3. Normalizarea datelor

Este procesul prin care se elimină redundanța datelor în proiectul bazei de date şi se construieşte un model de bază de date care susține diverse cerințe funcționale şi structuri alternative ale bazei de date. Sunt aplicate în normalizarea unei baze de date următoarele trei reguli.

N_1. Toate atributele trebuie specificate o singură dată. Aceasta este *forma intâi normală*.

N_2. Un atribut trebuie să depindă în întregime de identificatorul unic al entității pe care o descrie. Aceasta este *forma a doua normală.* Se pun atributele într-o tabelă în care depind exclusiv de o cheie principală. Nu se utilizează tabele în care atributele să nu depindă exclusiv de o singură cheie!

N_3. Pentru a fi în *forma normală a treia,* fiecare tabelă trebuie să posede o singură cheie primară, iar datele din tabelă trebuie să depindă exclusiv de cheia primară a tabelei.

3.4. Cerințele minimale de definire a SGBDR

Dintre primele versiuni apărute, practic nici un sistem de gestiune a bazelor de date nu reuşea să acopere în întregime regulile lui Codd. De aceea s-au formulat o serie de criterii minimale pe care trebuie să le satisfacă un sistem de gestiune a bazelor de date pentru a putea fi considerat relațional.

Pentru ca un SGBD să fie *minimal relațional*:

1. toate datele din cadrul bazei de date sunt reprezentate prin valori în tabele;
2. între tabele nu există pointeri observabili de către utilizatori – operațiile cu relații nu fac apel la pointeri, indecşi, fişiere inverse etc.;
3. sistemul suportă operatori relaționali de proiecție, selecție şi joncțiune natural, fără limitări impuse. Unitatea de informație în cadrul acestor operații trebuie să fie relația.

SGBD este *complet relațional* dacă este minimal relațional şi satisface în plus următoarele condiții:

– sistemul suportă toate operațiile de bază ale algebrei relaționale;

– sistemul suportă două dintre restricțiile de integritate de bază ale modelului relațional şi anume unicitatea cheii unei relații şi restricția referențială;

SGBD este *pseudorelațional* dacă satisface condițiile 1 şi 3 de mai sus;

SGBD este cu *interfață relațională* dacă satisface condițiile 1 şi 3 de mai sus şi condiția 3 este îndeplinită numai în raport cu operația de interogare.

4. Componentele funcționale ale sistemului Visual FoxPro

Sistemul de gestiune a bazelor de date este componenta unui sistem de baze de date, care are rolul de a permite descrierea şi manipularea datelor conform unui model de date. La acest moment, în lume, cea mai mare parte a SGBD-urilor utilizate sunt bazate pe modelul relațional.

Visual FoxPro este un SGBD complet relațional, cu limbaj propriu, care suportă un nucleu extins din limbajul relațional SQL. Varianta "Visual" de FoxPro (VFP) a ajuns la versiunea 6.0 lansată în 1998 în cadrul pachetului de programe Visual Studio 6.0, care mai conține următoarele programe: Visual C++, Visual Basic 6.0 şi Visual Java 6.0. VFP este un sistem rapid, modularizat, flexibil, care nu necesită resurse de calcul foarte mari, îmbină programarea procedurală (prin limbaj propriu) cu cea descriptivă, pe obiecte (programare vizuală).

Utilizatorii neinformaticieni au la dispoziție o gamă largă de generatoare pentru ecrane, meniuri, rapoarte etc.

Utilizatorii de specialitate (informaticieni) pot dezvolta programe în limbaj propriu şi în SQL, aplicații (prin componentele **Designer** şi **Application**).

Administratorul bazei de date are la dispoziție instrumente oferite de VFP pentru asigurarea securității şi integrității datelor, pentru refacerea bazei de date etc.

4.1. Programarea orientată pe obiecte

În programarea orientată pe obiecte, implementată în Visul FoxPro, se utilizează conceptele descrise în continuare.

– *Clasa de obiecte* – reprezintă un tip abstract de date care defineşte structura obiectelor din acea clasă (proprietățile) şi mulțimea de metode (operații) pentru obiectele respective.

O clasă este un model sau o schiță care defineşte caracteristicile unui obiect şi în care este descris modul în care trebuie să arate şi să se comporte obiectul. Se poate crea o clasă nouă utilizând comanda CREATE CLASS sau modulul de proiectare a clasei (**Class Designer**). Rezultă un fişier cu extensia *.vcx* (**Visual Class Library**) care va conține clasele definite de utilizator.

Subclass este folosit pentru definirea unei noi clase, pentru un obiect care are ca punct de plecare definirea din altă clasă (clasă părinte). Noua definire va moşteni orice modificare din clasa părinte. O subclasă poate avea toate funcționalitățile unei clase existente şi în plus orice control sau funcționalitate care se doreşte a fi inclusă. Dacă, de exemplu, clasa de bază este telefonul, se poate

construi o subclasă care are toate funcționalitățile telefonului original şi în plus se pot adăuga caracteristici speciale. Astfel, subclasele permit reutilizarea secvenței de instrucțiuni sursă (reutilizarea codului). Utilizarea subclaselor este una din căile de reducere a mărimii codului care trebuie scris. Se începe cu definirea unui obiect care este apropiat de cel dorit şi se gestionează.

– *Obiectele* –reprezintă o colecție de proprietăți care se referă la aceeaşi entitate. Aceste proprietăți descriu structura de date a obiectului. Un obiect are un nume, prin care este referit, un identificator unic, metode, o implementare (privată – utilizatorul nu are acces) şi o interfață (care este publică). Cererile adresate unui obiect pentru a returna o valoare sau pentru a schimba o stare se numesc *mesaje*.

Obiectul este o instanță a unei clase, care combină atât date, cât şi proceduri. De exemplu, un control într-un videoformat (**Form**) este un obiect.

Clasele şi obiectele sunt entități apropiate şi înrudite, dar nu se identifică în totatlitate. O clasă conține informații despre modul în care un obiect trebuie să arate şi să se comporte, dar nu este obiectul ca atare. O clasă este o schiță, o schemă a unui obiect.

Operatorii modelului orientat pe obiecte se referă la actualizarea metodelor, a proprietăților, a claselor, a instanțelor. La baza operațiilor dintr-un astfel de model stau mesajele, care ajută obiectele să comunice între ele.

Restricțiile de integritate a datelor:

- orice obiect respectă restricțiile impuse clasei din care face parte;
- identificatorul obiectului asigură integritatea referirii la acesta;
- accesul la obiect este limitat la folosirea protocolului de mesaje definit pentru clasa din care face parte.

– *Metoda* – defineşte operațiile permise (operatorii) asupra obiectului, adică defineşte comportamentul acestuia. Metoda este o acțiune pe care un obiect este capabil să o realizeze. De exemplu, obiectele de tip **List Boxes** au ataşte ca metode: **AddItem, RemoveItem**, **Clear** pentru a gestiona conținutul listelor.

– *Proprietatea* – este un atribut al unui control, câmp sau obiect dintr-o bază de date, pe care programatorul îl stabileşte pentru a defini una din caracteristicile obiectului sau un aspect al comportării acestuia. De exemplu, proprietatea **Visible** dacă este fixată pe **True**, face ca obiectul respectiv să fie vizibil la momentul rulării videoformatului. Proprietățile obiectului pot fi schimbate

în fereastra de *proprietăţi*, în general, în timpul proiectării videoformatului; există situaţii când anumite proprietăţi pot fi schimbate şi în momentul execuţiei videoformatului, ca urmare a apariţiei unor evenimente.

– *Evenimentul* – reprezintă o acţiune, recunoscută de un obiect pentru care se poate scrie o secvenţă de cod pentru răspuns. Evenimentele pot fi generate de o acţiune a utilizatorului, cum este apăsarea butonului stâng al mouse-ului (**Click** event) sau apăsarea unei taste (**Keypress** event), prin secvenţele de program sau de către sistem (*timers*).

Un obiect are anumite proprietăţi, de exemplu, un telefon are o anumită culoare şi mărime. Când instalăm un telefon pe birou, el are o anumită poziţie. Receptorul poate fi pus sau nu în furca. Obiectele create au de asemenea proprietăţi care sunt determinate de clasa din care face parte obiectul. Aceste proprietăţi pot fi stabilite în momentul proiectării sau la momentul execuţiei aplicaţiei. De exemplu, anumite proprietăţi pe care le poate avea un obiect de tip **Check Box** sunt enumerate mai jos:

Proprietate	**Descriere**
Caption	Textul descriptiv de pe lîngă Check Box
Enabled	Specifică dacă un obiect Check Box poate fi adresat de utilizator.
ForeColor	Culoarea textului stabilit prin Caption.
Left	Poziţia faţă de marginea din stânga a obiectului Check Box.
MousePointer	Forma pointer-ului când trecem pe deasupra obiectului cu mouse-ul.
Top	Poziţia faţă de marginea de sus a obiectului Check Box.
Visible	Specifică când un obiect Check Box este vizibil.

Fiecare obiect recunoaşte şi răspunde la anumite acţiuni numite *evenimente*. Un eveniment este o activitate specifică şi predeterminată iniţiată de utilizator sau de sistem. Evenimentele, în majoritatea cazurilor, sunt generate de interacţiunea cu utilizatorul. De exemplu, la telefon, un eveniment este declanşat în momentul în care un utilizator ridică receptorul din furcă. Evenimentele sunt declanşate şi atunci când utilizatotul apasă pe butoane pentru a forma un număr de apel.

În Visula FoxPro acţiunile utilizatorului care declanşează evenimente includ: apăsarea butonului stâng de la mouse (**Click**), mişcarea mouse-ului (**MouseMove**) şi apăsarea unei taste

(**Keypress**). Sistemul poate iniția evenimente în cazul în care întâlneşte o linie de cod care cauzează eroare, cum ar fi mesaje specifice şi chiar întreruperea execuției aplicației.
Metodele sunt proceduri care sunt asociate cu un obiect. Metodele sunt diferite de procedurile normale din Visual FoxPro: metodele sunt legate intrinsec de evenimente.

Evenimentele pot avea asociate metode. De exemplu, dacă se scrie un cod pentru o metodă pentru evenimentul **Click**, acel cod este executat atunci când are loc evenimentul **Click**. De asemenea, metodele pot exista independent de orice eveniment. Metodele trebuie să fie apelate explicit în cod. *Setul de evenimente este fix, nu se pot crea evenimente noi!*

Să considerăm câteva evenimente asociate cu un obiect de tip *check box*:

Eveniment	Descriere
Click	Utilizatorul face click pe check box.
GotFocus	Utilizatorul selectează check box cu mouse-ul sau apăsând tasta [TAB].
LostFocus	Utilizatorul selectează un alt obiect.

În tabelul de mai jos sunt date câteva metode asociate cu obiectul de tip *check box*:

Metodă	Descriere
Refresh	Valoarea din check box este actualizată pentru a reflecta orice modificare apărută în datele sursă.
SetFocus	Controlul este dat către check box ca şi cum utilizatorul ar apăsa tasta TAB pentru a selecta check box.

Toate proprietățile, evenimentele şi metode asociate unui obiect sunt specificate în definirea clasei.

Programarea orientată spre obiecte este un sistem de programare care permite abstractizarea şi organizarea modulară de tip ierarhie şi are caracteristici polimorfice, de moştenire şi încapsulare.

Polimorfismul – este abilitatea de a avea metode cu acelaşi nume, dar cu un conținut diferit pentru clasele înrudite. Procedura de utilizare este determinată în momentul execuției de către clasa unui obiect. De exemplu, obiectele înrudite pot avea amândouă metode *Draw*. O procedură transmite un astfel de obiect ca parametru poate apela metoda *Draw* fără a mai fi nevoie să ştie ce tip de obiect

este parametrul transmis. Polimorfismul reprezintă posibilitatea ca diferitele obiecte să poată răspunde diferit la aceleaşi mesaje.

Moştenirea – este un termen din programarea orientată spre obiecte şi reprezintă abilitatea unei subclase de a prelua caracteristicile clasei părinte (clasa de bază). Când caracteristicile clasei părinte se modifică, subclasa derivată moşteneşte noile modificări. De exemplu, dacă se adaugă o nouă proprietate *IsBold* la un control de editare, orice subclasă care are la bază clasa acestui control va avea de asemenea proprietatea *IsBold*. Asadar, moştenirea reprezintă capacitatea unui obiect de a-şi deriva datele şi funcționalitatea din alte obiecte.

Prin intermediul moştenirii dacă se face o modificare într-o clasă, modificarea se reflectă în toate subclasele care aparțin clasei respective. Această actualizare automată facută prin moştenire aduce beneficii în privința timpului şi a efortului de proiectare. De exemplu, dacă un producător de telefoane doreşte să schimbe modul de formare a numărului prin trecerea de la sistemul bazat pe disc la sistemul de telefon cu butoane, va economisi multă muncă dacă va fi capabil să facă modificările în schema principală şi dacă toate telefoanele realizate anterior pe baza acestei scheme principale vor moşteni automat modificările aduse comparativ cu situația în care ar trebui să facă modificările individual la toate telefoanele existente.

Dacă se descoperă o greşeală în clasa de bază, în loc să fie modificat codul pentru fiecare subclasă se remediază eroarea în clasa de bază şi noua modificare se propagă în toate subclasele aferente.

Încapsularea – în programarea orientată spre obiecte este legată de abilitatea de a conține şi ascunde informațiile despre un obiect, cum sunt cele referitoare la structura internă a datelor şi codul sursă. Încapsularea izolează complexitatea internă a modului de operare a obiectului de restul aplicației. De exemplu, când se fixează proprietatea *Caption* la un buton de comandă nu este necesară cunoaşterea modului de stocare a şirului de caractere. Se spune că obiectul din acest punct de vedere este transparent. Încapsularea permite ca descrierea obiectelor să se facă astfel încât să nu existe acces din afara obiectului la datele sale (“black box”).

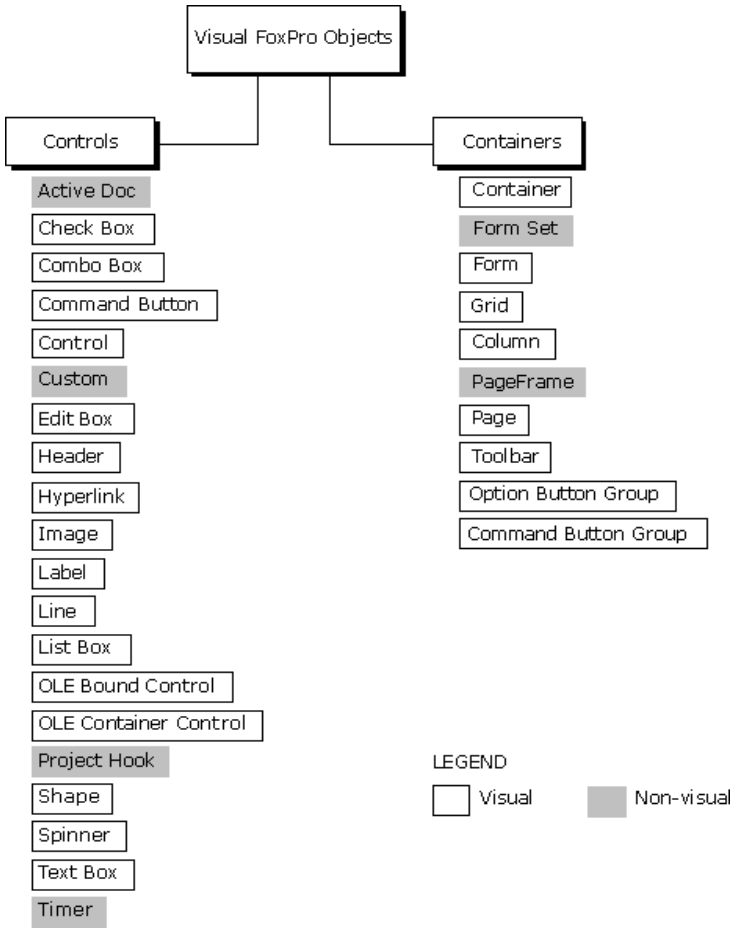
Abstractizare – este procesul de identificare a caracteristicilor distinctive a unei clase sau obiect fără a fi nevoie de a procesa toate informațiile referitoare la clasă sau obiect. Când se crează o clasă, de exemplu, un set de butoane pentru navigare într-o tabelă, setul se poate utiliza ca o singură entitate în loc de a ține evidența componentelor individuale (butoane) şi modul în care acestea interacționează.

Proiectarea şi programarea orientată spre obiecte reprezintă o schimbare a focalizarii activităţii faţă de programarea procedurală standard. În loc de a gândi schema logică şi algoritmul de la prima linie la ultima linie de cod sursă, trebuie să ne concentrăm asupra creării de obiecte autoconţinute într-o aplicaţie care să posede atât o funcţionalitate privată cât şi o funcţionalitate expusă utilizatorului. Între cele două tipuri de funcţionalitate există o interfaţă prin care utilizatorul poate schimba anumite proprietăţi, dar nu are acces la codul sursă intrinsec al obiectului.

În Visula FoxPro videoformatele (**Form**) şi controalele (**Control)** sunt obiecte care sunt incluse în aplicaţii. Aceste obiecte pot fi manipulate prin intermediul *proprietăţilor*, *evenimentelor* şi *metodelor*.

4.2. Ierarhia claselor în Visual FoxPro

La crearea unei noi clase este de mare ajutor cunoaşterea ierarhiei obiectelor din Visual FoxPro, preluată din [8]:



4.2. Arhitectura VFP 6.0

Arhitectura SGBD VFP 6.0 (Figura 1) corespunde unui model complet relațional, componentele sale fiind structurate pe trei niveluri: nucleul (*kernel*), interfețele (*interfaces*) şi instrumentele (*toolkit*).

1. *Nucleul* – este componenta centrală a sistemului. Din nucleu fac parte:

– *Limbajul FoxPro* care este propriu sistemului, este de tip procedural. El conține comenzi atât pentru descrierea datelor (LDD) cât şi pentru manipularea datelor (LMD). Tot aici sunt incluse comenzile pentru programarea vizuală din tehnologia orientată spre obiecte.

– *Nucleul extins SQL* – este un subset din standardul SQL. Acesta este un limbaj relațional descriptiv, care conține atât comenzi pentru descrierea datelor cât şi pentru manipularea datelor.

2. *Interfețele* – sunt produse VFP pentru dezvoltarea aplicațiilor cu baze de date relaționale. Ele au următoarele componente:

– **Designer** – permite crearea de diferite obiecte VFP: tabele (**Table**), cereri de regăsire (**Query**), videoformate (**Form**), rapoarte (**Report**), etichete (**Label**), meniuri (**Menu**).

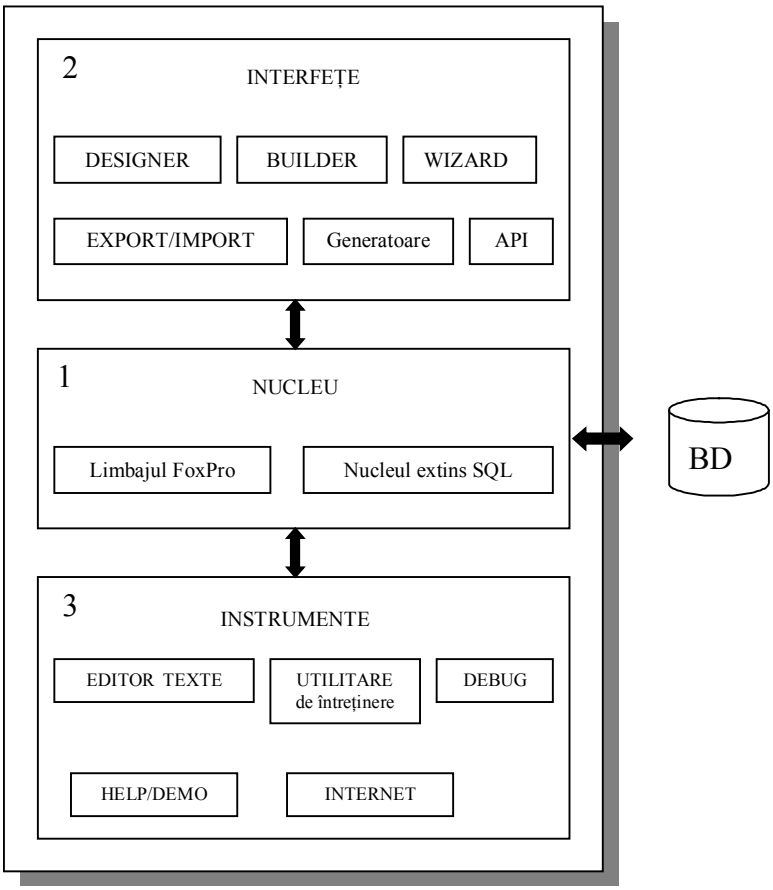


Fig. 1

– Un produs **Designer** se apelează printr-o comandă CRETE/MODIFY aferentă obiectului respectiv. Comanda poate fi generată de sistem sau scrisă de utilizator.

– **Builder** – permite adăgarea rapidă a unor noi elemente la obiectele tip VFP create deja cu produsul **Designer**.

– **Wizard** – permite realizarea completă şi rapidă a obiectelor de tip VFP. Utilizarea modulului **Wizard** presupune asistarea utilizatorului de către sistem în proiectarea obiectelor. Generarea lor se face automat, pe baza opțiunilor exprimate de utilizator, sub formă de răspuns la diferite întrebări. Acest modul face parte din categoria instrumentelor de proiectare asistată de calculator.

– **Export/Import** – permite schimbul de date între VFP şi alte sisteme.

– Generatoare specializate pentru realizarea proiectelor (PROJECT), a aplicațiilor (APP), a documentației (DOC).

– API (Application Programmer's Interface) – permite apelarea din aplicația VFP a unor rutine scrise în limbajul C sau limbaj de asamblare.

3. *Instrumentele* – sunt produse VFP pentru întreținerea şi exploatarea bazei de date. Instrumentele includ:

Editor de texte – permite încărcarea şi editarea programelor sursă (fişiere cu extensia .PRG), precum şi a fişierelor ASCII. Apelarea editorului se poate face prin comanda dată în fereatra *Command*:

MODIFY COMMAND <nume_fişier>

– UTILITARE de întreținere – permit gestiunea fişierelor, setarea unor parametri de lucru, activități desfăşurate de administratorul bazei de date.

– **debug** – permite depanarea interactivă a programelor scrise în FoxPro.

– **Help/Demo** – permite instruirea interactivă a utilizatorilor.

– **Internet** – permite utilizarea unor sevicii de Internet (mail, transfer de fişiere etc.).

4.3. Moduri de lucru în VFP

După intrarea în VFP utilizatorul poate lucra în două moduri: cu meniul sistem şi prin comenzi. Ecranul principal VFP conține o fereastră de tip Microsoft cu următoarele elemente (Figura 2):

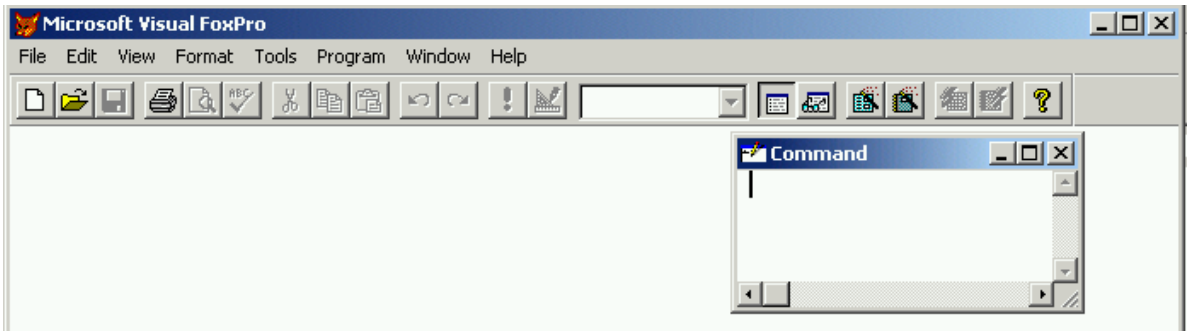


Fig. 2

– *bara de titlu* pe care este scris **Microsoft Visual FoxPro** (în stânga) şi butoanele pentru minimizare/maximizare şi închidere fereastră (dreapta).

– *bara meniului sistem* care conține un meniu orizontal cu opțiunile de lucru (modul de lucru meniu sistem): **File**, **Edit**, **View**, **Format**, **Tools**, **Program**, **Window**, **Help**. La selectarea unei opțiuni apare un submeniu vertical, de unde se alege mai departe subopțiunea dorită.

– *bara cu instrumente* care conține butoane (icon-uri) dispuse orizontal. Acestea pot fi active sau nu, în funcție de starea curentă de lucru. Prin aceste butoane se poate apela, sub o altă formă, o suboperațiune din meniul sistem.

– *fereastra de comandă* care conține un cursor şi permite introducerea unei comenzi VFP sau apelul unui program (modul de lucru prin comenzi).

– *aria de ieşire* este formată din restul spațiului neocupat din fereastra principală unde vor fi afişate rezultatele execuției unei comenzi sau a unui program VFP.

4.3.1. Modul de lucru meniu sistem

Este modul de lucru care permite apelarea tuturor instrumentelor şi interfețelor sistemului VFP. Soluția este adoptată de utilizatorii care preferă dezvoltarea aplicațiilor cu ajutorul generatoarelor. Efortul depus este redus şi nu se programează în cod sursă. Opțiunile din meniul principal ca şi cele din submeniuri pot fi apelate prin mouse, sau cu o combinație de taste. De exemplu, apelarea meniului **File** (Figura 3) se face tastând secvența [**ALT**] + [**F**] (litera subliniată din componența numelui meniului).

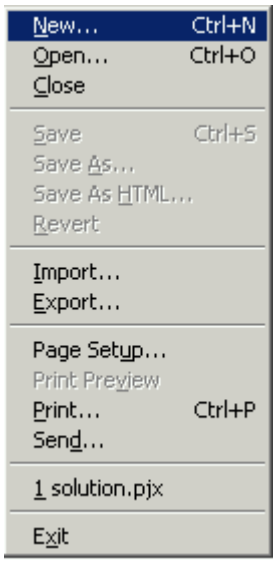


Fig. 3

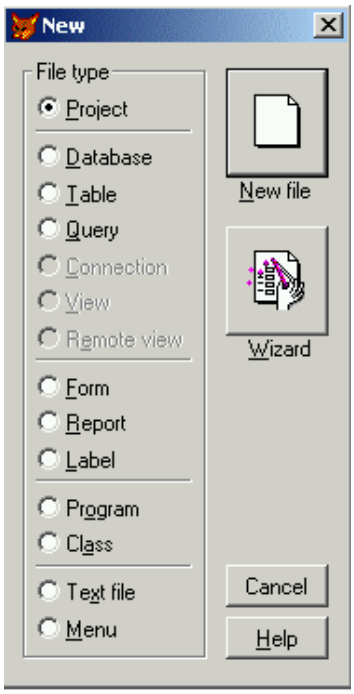


Fig. 4

Funcțiile opțiunilor din meniul principal al sistemului sunt:

– **File** – apelează instrumente pentru întreținerea fişierelor aferente unei baze de date VFP (Figura 3):

New – crează un obiect nou. Apare o listă din care putem selecta: **Project** (crearea unui proiect), **Database** (crearea de bază de date), **Table** (crearea unei tabele), **Query** (crearea unei cereri), **Connection** (realizează oconexiune), **View** (crearea unei viziune locală), **Remote View** (crearea unei viziune la distanță), **Form** (crearea unui videoformat), **Report** (crearea unui raport), **Label** (crearea etichetelor), **Program** (pentru crearea unui program sursă), **Class** (crearea unei clase de obiecte), **Text file** (crearea de fişier de tip text - ASCII), **Menu** (crearea unui meniu utilizator). În partea dreaptă există două butoane, pentru crearea unui fişier nou (**New**) şi pentru utilizarea asistentului (**Wizard**) (Figura 4).

Open (deschide) –apare o fereastră din care se alege tipul fişierului, directorul şi numele, cu opțiunile **New**, **Open**, **Cancel**.

Close – închide fişierul deschis.

Save/Save As – salvare, respectiv salvare cu redenumire.

Import/Export – permite importul respectiv exportul de date cu alte sisteme de gestiune a bazelor de date.

Print Preview – vizualizare înainte de ieşire la imprimantă.

Print – ieşire la imprimantă.

Send – trimite prin e-mail.

Exit – ieşire din program.

Edit – oferă facilităţi de lucru obişnuite într-o fereastră de editare de texte (Figura 5):

Undo/Redo – renuţă la ultima modificare în text / repetă ultima acţiune în text.

Cut/Copy/Paste – tăiere, memorare în memoria tampon, copiere la o nouă locaţie în fişier a unui text.

Clear – ştergere text.

Select all – selectează întreg textul.

Find/Find again/Replace – caută/caută în continuare un şir de caractere sau înlocuieşte şirul găsit cu un altul. Căutarea se poate face cu activarea/dezactivarea opţiunii de diferenţiere între litere mari/litere mici.

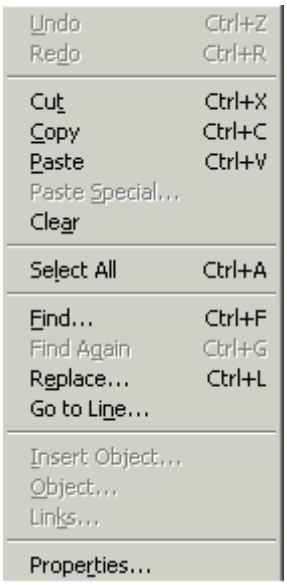


Fig. 5

– **View** – permite setarea unor parametri privind bara curentă de instrumente de lucru.

– **Format** – permite setarea unor parametri privind literele (Font), spaţierea etc.

– **Tools** – permite apelul interfeţelor şi instrumentelor VFP.

– **Program** – permite lansarea sau oprirea unuia din paşii ce se parcurg la execuţia unui program VFP (Figura 6).

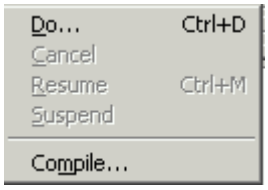


Fig. 6

Do – lansează în execuţie un program.

Cancel – anulează execuţia programului.

Resume – reia execuţia programului.

Suspend – suspendă execuţia programului.

Compile – compilează programul.

Window – permite setarea parametrilor ferestrei curente de lucru (inclusiv fereastra de comandă).

Help – apelează instrumentele pentru autodocumentare.

Datele pot fi introduse în două moduri, utilizând fie tabele independente (**File/New/Table**) care nu aparţin unei baze de date, fie creând o bază de date (**File/New/Database**), în care se pot introduce tabele existente sau se pot crea noi tabele.

Relațiile între tabele se pot stabili în ambele cazuri. Există trei tipuri de relații între datele unor tabele: $1 \leftrightarrow 1$, $1 \leftrightarrow n$, $m \leftrightarrow n$.

Deschiderea unei tabele individuale, cu secvența **File**/**Open**/**Table**/*nume_tabelă* nu duce automat la afişarea conținutului acesteia în spațiul de lucru. Pentru aceasta trebuie activat icon-ul care corespunde opțiunii **Data Session**, care va afişa într-o fereastră o serie de opțiuni legate de tabelă (Figura 7):

Properties – pentru modificarea/consultarea structurii tabelei (nume, tip, poziție câmp).

Browse – pentru afişare date/nume câmpuri.

Open – pentru deschidere altor tabele.

Close – închidere fişier selectat.

Relation – stabilirea relațiilor între tabele independente, deschise în sesiunea curentă.

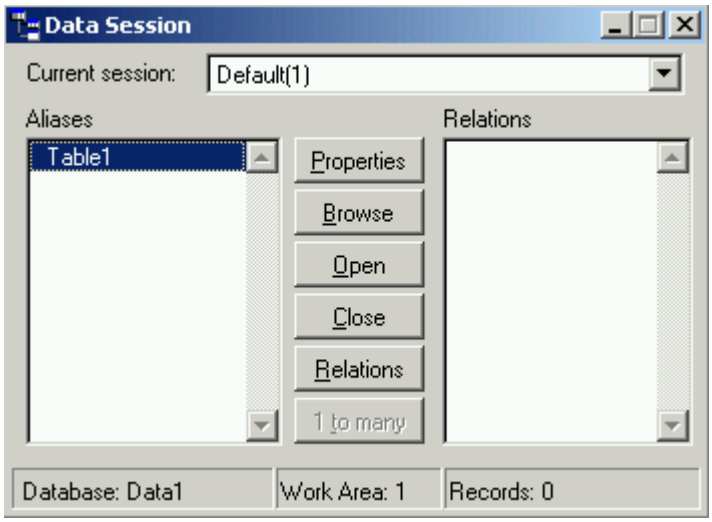


Fig. 7

Fig. 8

La alegerea opțiunii **Browse** (răsfoire), fie din meniul **View** fie din fereastra **Data Session**, va apare meniul **Table** în bara de meniuri cu următoarele subopțiuni (Figura 8):

Go to Record – regăsirea unei înregistrări după diferiți parametri: număr înregistrare, deplasare la începutul/sfârşitul tabelei, localizare înregistrare după valoarea unui câmp etc.

Append New Record – adăgarea unei noi înregistrări.

Append Records – adăugarea de noi înregistrări dintr-un alt fişier

Delete Records – marcare pentru ştergere (la nivel logic).

Recall Records – anularea marcării pentru ştergere.

Remove Deleted Records – ştergerea definitivă din tabelă a înregistrărilor marcate (la nivel fizic).

Replace Field – actualizarea câmpului unei înregistrări (schimbarea valorii câmpului).

Size Field – modificarea lăţimii de afişare a câmpului în browser.

Move Field – schimbarea poziţiei de afişare a câmpului în browser.

În cazul în care se construieşte o aplicaţie în care va fi folosită o gamă largă de obiecte din VFP, (baze de date, tabele independente, cereri, videoformate, rapoarte, etichete, programe sursă, clase de obiecte, meniuri utilizator, iconu-uri etc.) se va construi un proiect (**Project**), în secţiunile căruia se pot declara aceste obiecte (Figura 9). Întreg proiectul se va finaliza într-un program executabil care va conţine toate obiectele declarate ca fiind utilizate în aplicaţie. Pentru a realiza proiectul în formă executabilă, în prealabil se foloseşte opţiunea **Build**, pentru a realiza compilarea şi link-editarea (§12.2.2.).

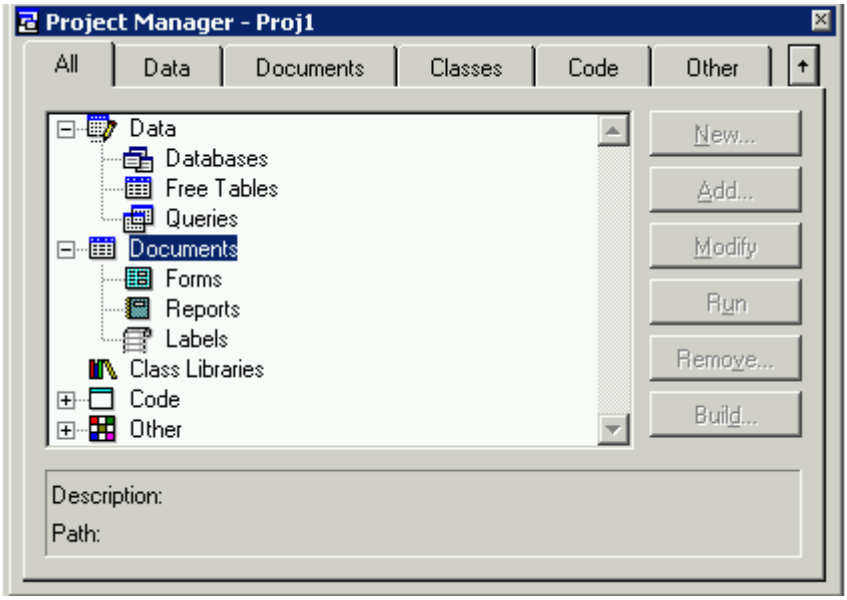


Fig. 9

4.3.2. Modul de lucru prin comenzi

Activitatea în acest mod de lucru se desfăşoară în fereastra de comandă din ecranul principal al VFP (**Command**). Aici se găseşte poziționat cursorul şi utilizatorul poate lucra în stil interpretor sau compilator.

5. Organizarea datelor înVisual FoxPro

5.1. Manipularea bazei de date şi a tabelelor în VFP

Manipularea presupune următoarele operaţii care pot fi executate fie în modul asistat, fie în modul comandă, fie în modul procedural:

– Crearea bazei de date/tabelei

– Deschiderea bazei de date/tabelei. Deschiderea unei tabele presupune rezervarea în memoria internă a calculatorului a unei zone, numită *zonă de lucru*, în care VFP va memora toate informaţiile necesare utilizării acestei tabele (structură, număr de înregistrări etc).

Într-o zonă de lucru, la un moment dat, este deschisă o singură tabelă, iar simultan pot fi deschise 32767 tabele. Identificarea zonelor de lucru de face printr-un număr natural cuprins între 1 şi 32767, iar primele 10 zone şi prin litere, *A-J*. La lansarea sistemului VFP, automat este selectată zona de lucru *A* (sau 1).

Cu comanda

SELECT *litera|cifra*

se poate selecta o altă zonă de lucru.

– Efectuarea operaţiilor de consultare, actualizare (adăgare, modificare, ştergere) asupra bazei de date/tabelei

– Închiderea bazei de date/tabelei

5.1.1. Crearea bazei de date

Sistemul VFP permite crearea unei baze de date şi introducerea tabelelor care vor face parte din baza de date:

– *definirea structurii* acestora;

– *crearea fişierului* (fişierelor) în care se pot stoca datele;

– *stabilirea relaţiilor* între tabelele bazei de date,

Crearea se face o singură dată, înainte de utilizare.

Notă Secvenţele din linia de comandă care apar între paranteze drepte au semnificaţia de element opţional.

Crearea unei baze de date se face cu comanda:

CREATE DATABASE [<*nume_bază_de_date*>]?

nume_bază_de_date – specifică numele bazei de date care va fi creată.
? – afişează o fereastră de dialog pentru specificarea numelui bazei de date.

Fişierul corespunzător bazei de date are extensia .DBC. La crearea bazei de date se face automat şi deschiderea ei. Adăugarea unei tabele în baza de date se face cu comanda ADD TABLE sau folosind bara de instrumente DATABASE DESIGNER.

5.1.2. Crearea tabelelor

Crearea tabelelor se realizează în funcție de modul de lucru, astfel:

1. *modul asistat* – se utilizează secvența de adresare a meniului sistem: **File/New** se alege opțiunea **Table** din fereastra de dialog şi se apasă butonul **New**. Această secvență are ca efect apariția unei ferestre de dialog **Create**, în care se specifică discul logic, directorul şi numele tabelei care va fi creată. Deplasarea până la poziția dorită în fereastră, se face utilizând tasta [**TAB**] sau mouse-ul. După introducerea numelui tabelei se apasă butonul **Save** (fişierul va fi salvat cu extensia .DBF) şi se afişează o nouă fereastră **Table Designer**. În acestă fereastră, se selectează opțiunea **Fields** afişată sus, utilizatorul trebuie să introducă structura tabelei. Fereastra conține următoarele câmpuri de preluare a informațiilor structurale: **Name**, **Type**, **Width**, **Dec, Index**, **NULL** şi patru butoane pentru opțiunile: **OK**, **Cancel**, **Insert**, **Delete**. Sub **Name** se tastează numele câmpului, format din maxim 10 caractere, începând cu o literă, iar sub **Type** se declară tipul câmpului nou creat sau se alege o opțiune din meniul pop-up afişat.

Câmpul poate fi de tip:

– *caracter* – admite un şir de maxim 254 caractere (caractere alfanumerice şi spații goale);
– *numeric* – admite maxim 20 de caractere, numere, caracterele +/-, punctul zecimal;
– *logic* – admite maxim un caracter (**T** sau **Y**, **F** sau **N**);
– *dată* – admite maxim 8 caractere şi are formatul implicit *ll/zz/aa*. Formatul poate fi schimbat cu comanda SET DATE TO (de exemplu BRITISH şi are ca efect schimbarea în formatul *zz/ll/aa)*;
– *memo* – admite implicit 10 caractere, însă sistemul poate stoca blocuri mari de text pentru fiecare înregistrare. Dimensiunea blocurilor de text este limitată de spațiul pe hard disk. Tabelele care au declarate în structură câmpuri de tip memo sunt stocate sub forma a două fişiere, unul cu extensia .DBF şi celălalt cu extensia .FTP în care se salvează câmpurile memo ale tabelei.
– *general* – se foloseşte pentru memorarea elementelor de tip OLE (Object Linking and Embededing), texte, foi de calcul tabelar (Excel), imagini, sunete etc.

Notă: Introducerea datelor în câmpurile memo şi general este diferită de introducerea datelor în alte câmpuri. Se poziționează cursorul pe câmpul memo, se apasă simultan tastele [**Ctrl**]+[**PgDown**] şi apare o fereastră cu acelaşi nume ca al câmpului memo în care poate fi introdus textul. Textele introduse se salvează apăsând tastele [**Ctrl**]+[**W**], fereastra memo se închide şi cursorul revine în fereastra de adăugare a datelor. Cu **[Ctrl]+[Esc]** se iese din fereastra memo fără salvarea datelor introduse în fişierul memo.

Width – opțiune care specifică numărul maxim de poziții ale câmpului.
Dec – specifică mărimea părții zecimale, în cazul câmpurilor numerice.
Index – specifică dacă atributul (câmpul) respectiv este index (ascendent/descendent) sau nu (§8.2.).
NULL – specifică dacă atributul respectiv admite sau nu valori de tip **NULL** (cu semnificația unor date lipsă sau inaplicabile).

Dacă se doreşte schimbarea poziției unui câmp în structură, se apasă tasta [**Tab**] până când cursorul se pozționează pe câmpul respectiv şi se apasă simultan tastele [**Ctrl**] + [**PgDown**], [**Ctrl**] + [**PageUp**]. Aceasta va avea ca efect deplasarea câmpului selectat cu o poziție în jos, sau în sus.

Operația se poate efectua şi cu mouse-ul: se selectează rândul care conține câmpul a cărei poziție se modifică, cursorul de poziție se mută pe acest rând; ținând apăsat butonul stâng de la mouse pe cursor şi deplasând mouse-ul sus/jos se va muta câmpul selectat, în cadrul structurii generale.

Butonul cu opțiunea **Insert** se foloseşte pentru adăugarea unui nou câmp.

Butonul cu opțiunea **Delete** se foloseşte pentru ştergerea unui câmp (cel marcat de cursor).

Butonul **Cancel** anulează modificările făcute, iar butonul **OK** validează modificările (sau folosind combinația de taste [**Ctrl**] +[**Enter**]).

După ce structura este salvată, fişierul fiind nou creat, apare un mesaj:

Input data records now? (**Y**/**N**)_

Dacă se răspunde cu **Y** este afişat un ecran în care se introduc datele în ordinea stabilită în structură.

5.1.3. Validarea câmpurilor unei înregistrări la introducere

În exemplul care urmează se crează o bază nouă de date GESTIUNE.DBC (dbc - data base container), Figura 10, care va conține tabelele: FURNIZOR.DBF şi MATERIALE.DBF cu următoarele câmpuri:

FURNIZOR.DBF	
Cod_furn	N(5)
Denumire	C(20)
Adresa	Memo
Cod_mat	N(5)
Cant	N(7)
Data_livr	D

MATERIALE.DBF	
Cod_mat	N(5)
Denumire	C(20)
Data_exp	D
UM	C(5)
PU	N(5)

După definirea structurii înregistrărilor în tabela FURNIZOR.DBF se selectrază câmpul *cod_furn* şi se completează cele trei câmpuri **Rule**, **Message**, **Default value** (Figura 11), astfel:

- la opțiunea **Rule** se introduce condiția logică de filtrare pentru acceptarea valorii înscrise în câmp. În acest caz se consideră că pentru câmpul *cod_furn* valorile acceptate sunt între 1 şi 5000. Condiția este:

  cod_furn<=5000 **and** *cod_furn*>=1

- la opțiunea **Message** se introduce între apostrof mesajul:

  '*Cod furnizor eronat !*'

- la opțiune **Default Value** se introduce valoarea 1. Dacă se omite declararea unei valori implicite, la introducerea datelor pentru câmpul *cod_furn*, se va afişa mesajul de eroare la orice încercare de introducere a datelor.

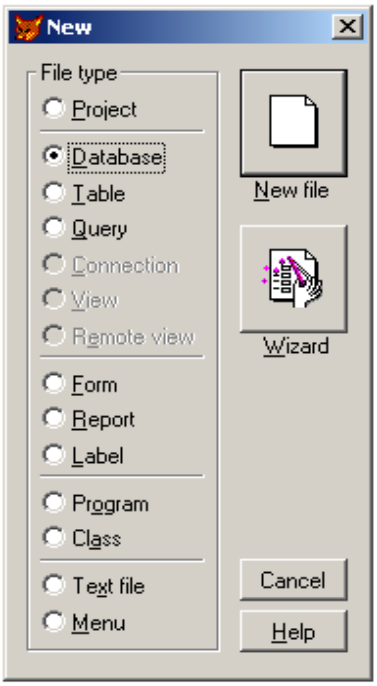


Fig. 10

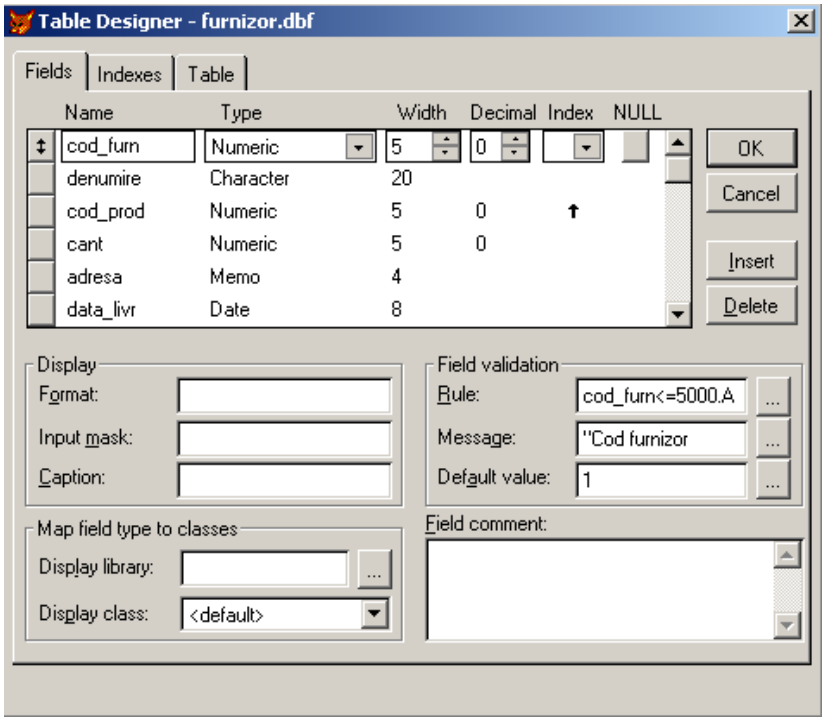


Fig. 11

După definirea structurii înregistrărilor în tabela MATERIALE.DBF se selectrază câmpul *um* şi se completează cele trei câmpuri **Rule**, **Message**, **Default Value** (Figura 12), astfel:

- la opțiunea **Rule** se introduce condiția logică de filtrare pentru acceptarea valorii înscrise în câmp. În acest caz se consideră că pentru câmpul *um* valorile acceptate sunt doar trei: *kg, ml, buc.* Condiția este:

  um='kg' **or** *um*='ml' **or** *um*='buc'

- la opțiunea **Message** se introduce între apostrof mesajul:
  '*Eroare unitate de masura !!*'
- la opțiune **Default Value** se introduce între apostrof valoarea '*buc*'. Dacă nu se declară nimic, practic, nu se pot introduce datele la fel ca mai sus.

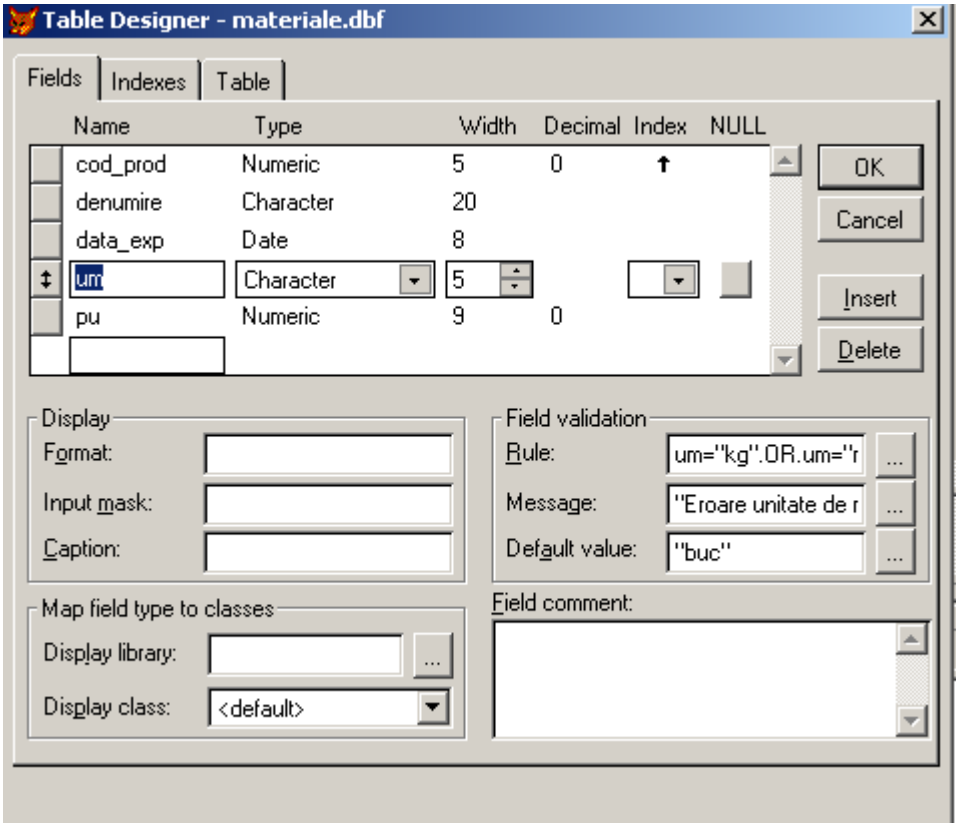


Fig. 12

2. ***modul comandă***: implică utilizarea comenzii CREATE TABLE din nucleul SQL:

CREATE TABLE *nume_tabelă* [FREE]

(*nume_câmp*1 *tip_câmp*1 [(*lungime_câmp* [,*precizie*])]

[,*nume_câmp*2…]

. . . . . .

)

unde:

- *nume_tabelă* – numele tabelei create;
- FREE – specifică că tabela nu va fi adăugat la baza de date deschisă;
- *nume_câmp*1 *tip_câmp*1 (*lungime_câmp* ,*precizie*) – specifică numele câmpului 1, tipul câmpului 1, - lungimea maximă şi numărul de zecimale pentru câmpurile numerice. O tabelă poate conține până la 255 de câmpuri. Dacă unul sau mai multe câmpuri acceptă şi valoarea **NULL** atunci numărul de câmpuri se reduce cu 1 (254).

Tipul câmpului se declară cu una din următoarele litere:

Tip_câmp	**Lungime_câmp**	**Precizie**	**Descriere**
C	*n*	–	câmp tip caracter de lungime *n*

D	–	–	câmp de tip dată calendaristică
T	–	–	câmp de tip timp (*hh/mm/ss*)
N	*n*	*d*	câmp numeric de lungime *n* cu *d* zecimale
F	*n*	*d*	câmp numeric în virgulă mobilă de lungime *n* cu *d* zecimale
I	–	–	câmp numeric întreg
B	–	d	câmp numeric dublă precizie
Y	–	–	moneda
L	–	–	câmp de tip logic
M	–	–	câmp de tip memo
G	–	–	câmp de tip general

Exemplu. Folosind modul comandă se crează o tabelă cu evidența studenților, cu numele ESTUD, cu următoarele câmpuri:

- număr matricol: MATR – de tip numeric, lungime maximă 5
- nume şi prenume: NUME – de tip caracter, lungime maximă 40
- adresă: ADRESA – de tip caracter, lungime maximă 30
- anul de studii: ANS – de tip numeric, lungime maximă 1
- anul intrării în facultate: ANIF – de tip numeric, lungime maximă 4
- data naşterii: DATAN – de tip dată.

Comanda:

CREATE TABLE ESTUD (MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), ANIF N(4), DATAN D)

Alte comenzi utilizate în crearea tabelelor

Pentru crearea unei tabele se pot utiliza şi comenzile COPY STRUCTURE EXTENDED TO sau CREATE ... FROM ...cu sintaxa

COPY STRUCTURE EXTENDED TO <*nume_tabelă*> [FIELDS<*listă_câmpuri*>] [DATABASES <*nume_bază_de_date*> [NAME] <*b_nume_tabelă*>]

unde:

- *nume_tabelă* – numele tabelei nou create;
- FIELDS <*listă_câmpuri*> – specifică câmpurile care urmează să apară în noua tabelă. Dacă nu se specifică nimic, implicit sunt luate toate câmpurile din tabelă sursă;
- DATABASE <*nume_bază_de_date*> – numele unei bazei de date existente la care va fi adăgată noua tabelă;
- NAME <*b_nume_tabelă*> – specifică numele tabelei aşa cum apare în baza de date.

Fişierul de structură creat are următoarele câmpuri:

FIELD_NAME	câmpul nume (10 caractere)
FIELD_TYPE	câmpul tip (maxim 1 caracter): **C**, **N**, **F**, **D**, **L**, **M**, **G**
FIELD_LEN	lungimea câmpului (maxim 3 caractere)
FIELD_DEC	număr de zecimale admis (maxim 3 caractere)

Comanda CREATE... FROM...crează o tabelă dintr-un fişier de structură (realizat cu comanda anterioară COPY STRUCTURE EXTENDED TO...). Sintaxa comenzii:

CREATE [<*nume_tabela*_1>] FROM <*nume_tabela*_2>

*nume_tabela*_1 – numele tabelei care va fi creată;

*nume_tabela*_2 – numele fişierului care conține structura care va fi utilizată la creare.

Exemplu. Folosind tabela creată anterior (ESTUD), să se realizeze o nouă tabelă (ESTUD1), care să conțină suplimentar un câmp numeric (lungimea maximă 5 caractere, din care 2 la partea zecimală) cu media de la bacalaureat (MBAC). Secvența de comenzi:

```
SELECT 2                          && selectează zona de lucru 2
USE ESTUD                         && se deschide tabela ESTUD în zona de lucru 2
COPY STRUCTURE  EXTENDED TO       && copiază structura tabelei ESTUD în fişierul TEMP
TEMP
USE TEMP                          && se deschide tabela TEMP
APPEND BLANK                      && se adaugă o înregistrare goală
REPLACE FIELD_NAME WITH           && se adaugă un nou câmp MBAC
"MBAC"
REPLACE FIELD_TYPE WITH "N"       && tip numeric
```

REPLACE FIELD_LEN WITH 5	&& de lungime maxim 5 poziții
REPLACE FIELD_DEC WITH 2	&& 2 poziții pentru partea zecimală
	&& la fiecare REPLACE se afişează mesajul "1 replacements"
CREATE ESTUD1 FROM TEMP	&& se crează tabela ESTUD1 cu structura dată de fişierul TEMP
USE ESTUD1	&& se deschide tabela ESTUD1
APPEND FROM ESTUD	&& se adaugă înregistrările din tabela inițială ESTUD
BROWSE	&& afişarea conținutului tabelei ESTUD1 în browser
CLOSE DATABASES	&& se închid toate tabelele
DELETE FILE TEMP.DBF	&& şterge fişierul TEMP.DBF

5.2. Deschiderea bazei de date/tabelei

Deschiderea bazei de date se face cu comanda:

OPEN DATABASE [<*nume_bd*>] [EXCLUSIVE | SHARED] [NOUPDATE]

unde:

nume_bd – numele bazei de date care trebuie deschisă;

EXCLUSIVE – alți utilizatori nu pot să acceseze această bază de date;

SHARED – deschide baza de date în modul partajat, ceilalți utilizatori au acces;

NOUPDATE – specifică faptul că nu se pot face schimbări în baza de date (deschis doar pentru citire).

Comanda pentru deschiderea unei tabele într-o zonă de lucru:

USE <*nume_fişier*> IN <*nr_zonă_de_lucru*> INDEX <*listă_fişiere_index*> ALIAS <*alias_tabelă*> [EXCLUSIVE | SHARED] [NOUPDATE]

unde:

nume_fişier – numele tabelei pe care vrem să o deschidem.

nr_zonă_de_lucru – numărul zonei de lucru asociate la deschiderea tabelei (numere 0, 1, 2.. sau litere A, B...).

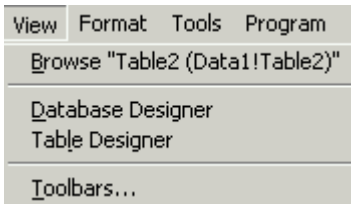
INDEX <*listă_fişiere_index*> - specifică un set de indecşi care se deschid odată cu tabela. Dacă tabela are un fişier de index structural compus, fişierul de index este deschis automat odată cu

fişierul. Lista de fişiere index conţine orice fel de nume de fişier de index, simplu (.IDX) sau nestructural compus (.CDX), separate prin virgulă, fără specificarea extensiei. Primul fişier de index din listă este cel care controlează accesarea şi afişarea datelor din tabelă (§8.2.).
ALIAS <*alias_tabelă*> - crează un nou nume pentru tabelă, diferit de numele extern. Referirea la fişier se poate face prin intermediul alias-ului, în funcţii sau comenzi care cer sau suportă alias-ul.
Specificarea zonei de lucru poate fi făcută şi separat asociind comanda SELECT la comanda USE.

SELECT <*nr_zonă_de_lucru*> | <*nume_alias*>
USE <*nume_fişier*>

Notă Semnul | în sintaxa unei comenzi marchează alternanţa unor clauze care se exclud.

În modul asistat, deschiderea se face utilizând meniul **File/Open**, se selectează **Database** sau **Table** din meniul pop-up afişat şi se specifică discul logic, subdirectorul şi numele bazei de date sau tabelei, precum şi modul de deschidere (se bifează sau nu, opţiunea **Exclusive**). Când toate aceste informaţii sunt specificate se apasă butonul **OK**.
După operaţia de deschidere în meniul **View** avem subopţiunile **Browse** (vizualizare conţinut), **Database Designer** (proiectantul pentru baza de date), **Table Designer** (proiectantul pentru tabele). Se pot face consultări cu **Browse**, modificări ale structurii cu **Database Designer** (adăugare, eliminare de tabele din bază, stabilirea/anularea relaţiilor între tabele) sau **Table Designer** (actualizare câmpuri).



5.3. Consultarea şi modificarea bazei de date/tabelei

5.3.1. Modificarea structurii de date a tabelelor în modul asistat

Modificarea structurii logice a unei tabele se poate face fie cu ajutorul comenzii MODIFY STRUCTURE din fereastra de comandă, fie activând butonul **Data Session** din bara de instrumente

În primul caz, după lansarea comenzii MODIFY STRUCTURE, se va afişa fereastra de dialog **Open**. Cu ajutorul săgeților direcționale se poziționează cursorul pe tabela dorită a fi modificată. Se apasă tasta **[Enter]** şi va apărea fereastra de dialog **Structure**, utilizată şi la definirea structurii tabelelor.

În cel de al doilea caz, din fereastra **Data Session** se alege opțiunea **Open**, care deschide fereastra de selectare a discului logic, a subdirectorului şi a numelui de fişier. Se selectează fişierul şi se apasă butonul **OK**. În fereastra **Data Session** va fi afişat numele tabelei deschise. Pot fi deschise mai multe tabele simultan, numele lor fiind afişat în fereastra **Data Session**. Se alege opțiunea **Properties** care va activa fereastra de dialog **Structure**.

În ambele cazuri operația **Open** se va face cu opțiunea **Exclusive** pentru a putea face modificări.

În fereastra **Structure**, poziționarea cursorului se face apăsând tasta **[Tab]** pentru deplasare înainte sau **[Shift]** + **[Tab]** pentru deplasare înapoi, în zona ce cuprinde descrierea structurii, unde vrem să inserăm un nou câmp. În partea dreaptă a ferestrei vor fi active opțiunile **Insert**, **Delete** (adăgare/ştergere). Se apasă tasta **[Insert]** sau butonul cu opțiunea **Insert** şi va fi afişată următoarea linie:

New field	character	10

Câmpurile ce urmează vor fi împinse cu un rând în jos. Se introduce numele câmpului în prima căsuță, se alege tipul şi lungimea maximă din meniurile pop-up. Pentru ştergere se poziționează cursorul pe câmpul respectiv şi se apasă tasta [**Delete]** sau se foloseşte butonul cu opțiunea **Delete]**. Salvarea modificărilor se face selectând opțiunea **OK** (buton), sistemul afişând următorul mesaj:
Make structure changes permanent ?

YES	NO

Opțiunea **YES** salvează noua structură a tabelei, fereastra de dialog **Structure** se închide şi controlul (cursorul) trece în fereastra de comandă.

Notă Modificarea structurii tabelelor se poate face şi cu comanda ALTER TABLE din nucleul SQL.

La alegerea opțiunii **Browse** (răsfoire), fie din meniul **View** fie din fereastra **Data Session**, va apare opțiunea **Table** în bara de meniuri cu următoarele subopțiuni:

Go to Record – regăsirea unei înregistrări după diferiți parametri: număr articol, deplasare la începutul/sfârşitul tabelei, localizare înregistrare după valoarea unui câmp etc.
Append New Record – adăgarea unei noi înregistrări.
Append Records – adăugarea de noi înregistrări dintr-un alt fişier
Delete Records – marcare pentru ştergere (la nivel logic).
Recall Records – anularea marcării pentru ştergere.
Remove Deleted Records – ştergerea definitivă din tabelă a înregistrărilor marcate (la nivel fizic).
Replace Field – actualizare câmp înregistrare (schimbarea valorii câmpului).
Size Field – modificarea lățimii de afişare a câmpului în browser.
Move Field – schimbarea poziției de afişare a câmpului în browser.

Vizualizarea datelor cu opțiunea **Browse**, se face într-o fereastră de tip **Windows** care cuprinde o tabelă, coloanele reprezentând câmpurile, iar liniile reprezentând înregistrările. Trecerea de la un câmp la altul se face apăsând tasta **[Tab] (**înainte), **[Shift]** + **[Tab] (**înapoi); pe verticală deplasarea se face cu săgețile **[Sageata sus]**, **[Sageata jos]**. Închiderea ferestrei **Browse** se face apăsând simultan tastele **[Ctrl]** + **[End]**.

5.3.2. Deplasări în tabelă. Căutări secvențiale

Odată cu deschiderea tabelei, acesteia i se asociază o locație de memorie în care este stocat numărul înregistrării curente (pointer-ul de înregistrare).

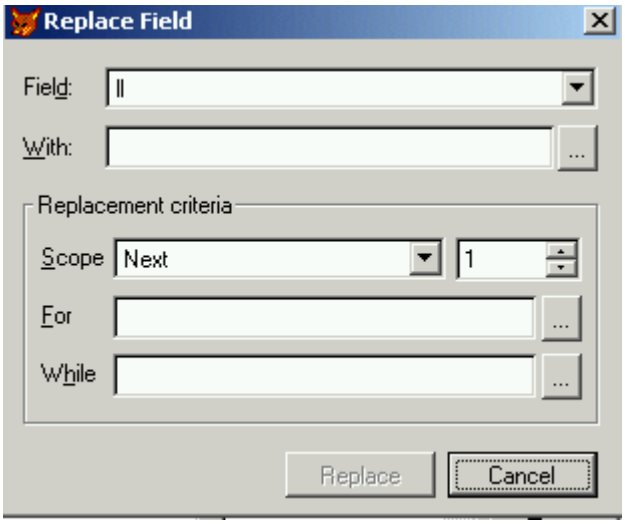
Există 2 tipuri de deplasări:

- criteriul utilizat este numărul înregistrării, care se realizează utilizând din submeniul **Go to Record** subopțiunile: **Top**, **Bottom**, **Next**, **Previous**, **Record** *n* (prima înregistrare, ultima înregistrare, următoarea înregistrare, înregistrarea anterioară, la înregistrarea *n*).
- criteriul utilizat se obține pe baza specificării unei condiții de tip **For** sau **While** aplicată unui câmp. Se utilizeză opțiunea **Locate** din submeniul **Go to Record**.

Pentru a modifica o înregistrare din tabelă se alege opțiunea **Browse** şi apoi din meniul **Table**, subopțiunea **Replace Field** din meniul sistem. Apare o fereastră de dialog (**Replace Field**) cu următoarele specificații:
Field - numele câmpului ale cărui valori vrem să le modificăm;
With – noua valoare.

Scope – domeniul (**Next** *n*, **All**, **Record** *n*, **Rest**): toate înregistrările, următoarele *n* înregistrări, numai pentru înregistrarea *n*, restul înregistrărilor inclusiv cea curentă pe care este poziționat cursorul.

For / **While** – vor fi modificate doar înregistrările care satisfac condiția. **For** este o condiție construită cu **Expression Builder** (constructorul de expresii) şi vor fi modificate acele înregistrări care au valoarea **True** pentru expresia dată. Clauza **While** este de asemenea o expresie logică şi înregistrările selectate trebuie să verifice condiția pentru a se realiza modificarea.



Constructorul de expresii are o regiune de editare în care vor fi introduse condițiile, manual sau asistat de calculator.

Prin apăsarea butonului **Replace** are loc operația de modificare.

5.4. Închiderea bazei de date/tabelei

Cu comanda CLOSE se închid diferite tipuri de fişiere:

CLOSE DATABASES [ALL] – închide baza de date curentă şi tabelele sale. Dacă nu este deschisă nici o bază de date, vor fi închise toate tabelele libere, fişierele de index, din toate zonele de lucru şi se va selecta zona de lucru 1. Clauza ALL specifică că vor fi închise toate bazele de date şi tabelele lor, toate tabelele libere şi toate fişierele de index.

CLOSE INDEX – închide toate fişierele de index pentru zona de lucru curentă (fişiere .IDX şi nestructurale .CDX). Un fişier de index structural (.CDX, compus – care se deschide automat cu tabela) nu va fi închis.

CLOSE ALL – închide toate bazele de date şi tabelele lor, toate tabelele libere, toate fişierele de index din toate zonele de lucru şi se selectează zona de lucru 1. Comanda nu va închide fereastra de comenzi, fereastra **Help**, fereastra **Debug**.

CLOSE TABLES [ALL] – închide toate tabelele din baza de date curentă, sau toate tabelele libere din toate zonele de lucru dacă nu există o bază de date deschisă. Includerea clauzei ALL închide toate tabelele din toate bazele de date şi tabelele libere, dar bazele de date rămân deschise.

6. Utilizarea asistentului Wizard în VFP

VFP este însoţit de peste 20 de programe de tip **Wizard**, care asigură funcţia de proiectare asistată de calculator. Acestea sunt programe interactive care ajută la crearea de videoformate, interogări/cereri (**Query**), importarea şi redimensionarea datelor, grafice, rapoarte cu date din diferite tabele, baze de date, tabele, etichete, viziuni, kit-uri de instalare, documentaţie, aplicaţii, documente pentru pagini de WEB. Programul **Wizard** are forma unei serii de ecrane, în care utilizatorul răspunde la întrebări sau alege o serie de opţiuni, în urma cărora se va realiza o operaţie sau se va genera un fişier. Lansarea programului **Wizard** se poate face:
Când: se alege opţiunea **New** din meniul sistem **File** sau selectând submeniului **Wizards** din meniul sistem **Tools**.

6.1. Table Wizard

Crează tabele bazate pe structura tipică a tabelelor. Asistentul pentru tabele pune la dispoziţia utilizatorului modele de tabele, acesta alegând pe cel care răspunde necesităţilor sale. Se poate modifica structura tabelelor şi câmpurilor, fie în derularea etapelor din **Wizard** sau mai târziu după ce asistentul a salvat fişierul. Dacă sunt deschise una sau mai multe baze de date, asistentul adaugă automat noua tabelă la baza curentă. Dacă nu este deschisă nici o bază de date asistentul crează tabele libere (*free tables*).

Lansarea asistentului se face din submeniul **Wizards** din meniul sistem **Tools**, opţiunea **Table**: în următorii paşi

pasul 1 – selectarea câmpurilor: din fereastra **Sample tables** se pot alege una sau mai multe tabele prin opţiunea **Add**. Pentru fiecare tabelă sunt afişate câmpurile disponibile (**Available Fields**) care pot fi adăugate în selecţia câmpurilor utilizatorului (**Selected Fields**) folosind butoanele: - adaugă câmpul selectat, - adaugă toate câmpurile tabelei, - elimină câmpul selectat - elimină toate câmpurile selectate.

pasul 2 – selectarea unei baze de date: tabela creată poate fi de sine stătătoare sau poate fi adăugată la o bază de date. Dacă se crează tabela într-o bază de date asistentul va furniza opţiunile de formatare în pasul următor.

pasul 3 – modificarea setărilor câmpurilor: se pot schimba setările câmpurilor pentru tabela creată. De exemplu, se poate modifica lungimea maximă a unui câmp de tip caracter.

pasul 4 – indexarea tabelei : se selectează câmpul care va fi cheie primară de indexare în noua tabelă. Se pot desemna de asemenea alte câmpuri drept chei secundare.

pasul 5 – setarea relațiilor: dacă tabela aparține unei baze de date, se pot stabili relațiile între câmpuri, între tabela nou creată şi tabelele existente în bază. Pentru definirea relației se face click cu mouse-ul pe butonul radio care identifică tipul relației, se selectează câmpul care se leagă în baza de date (dacă se selectează **New Field**, se va tipări numele) şi operația se termină cu click pe butonul **OK**.

pasul 6 – terminare (butonul **Finish**): noua tabelă va fi afişată într-o fereastră de tip **Browse**. Se poate alege opțiunea **Append New Record** din meniul sistem **Table**, pentru a adăuga înregistrări noi în tabelă. După salvarea fişierului, acesta poate fi deschis mai târziu pentru a modifica structura cu aplicația **Table Designer**.

6.2. Form Wizard

Form Wizard – crează un videoformat pentru date extrase dintr-o singură tabelă. Lansarea asistentului se face utilizând meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Form**, în următorii paşi:

pasul 1 – selectarea câmpurilor: se face pe baza opțiunilor utilizatorului exprimate în trei ferestre: **Databases and Tables**, **Available Fields**, **Selected Fields**, la fel ca în cazul **Wizard Table**.

pasul 2 – selectarea stilui videoformatului – specifică stilul controalelor din videoformat. În fereastra **Style** este afişată o listă cu tipurile de videoformate: **Standard**, **Chiseled**, **Shadowed**, **Boxed**, **Embossed**, **Fancy** etc. Se poate alege din panoul cu butoane radio (**Button Type**) tipul butoanelor utilizate: **Text Buttons**, **Picture Buttons**, **No Buttons**, **Custom Bbuttons**. Tipul de butoane ales va genera în videoformat, butoane pentru navigare în fişier (**Top**, **Previous**, **Next**, **Bottom**), **Find** (afişează cutia de dialog **Search** - caută), **Print** (tipăreşte un raport), **Add** (adaugă o nouă înregistrare), **Delete** (şterge înregistrarea curentă), **Exit** (închide videoformatul).

pasul 3 – sortarea înregistrărilor: se vor selecta acele cîmpuri care dorim să fie sortate. Se poate alege de asemenea o etichetă (intrare) pentru un index (TAG).

pasul 4 – terminare (butonul **Finish**): dacă s-a ales un număr mare de câmpuri la pasul 1 şi pentru a fi siguri că toate câmpurile vor încăpea în videoformat se poate alege opțiunea **Add pages for fields that do not fit**. În caz contrar, dacă numărul câmpurilor depăşeşte dimensiunea videoformatului, VFP va afişa formatul cu bare de deplasare (**Scroll Bars**). Videoformatul poate fi vizualizat înainte de salvare cu opțiunea **Preview**. După salvarea videoformatului, acesta poate fi deschis mai târziu şi modificat cu aplicația **Form Designer**.

6.3. Report Wizard

Crează rapoarte utilizând o singură tabelă liberă sau o viziune într-o bază de date. Asistentul pune întrebări într-o succesiune de paşi, prin care se specifică sursa şi câmpurile ce vor fi utilizate pentru a crea controalele raportului. Pentru lansarea asistentului, se alege din meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Report**. Din caseta de dialog **Wizard Selection** se alege **Report Wizard** şi se urmează paşii:

pasul 1 – selectarea câmpurilor: se introduc opțiunile în cele trei ferestre de dialog: **Databases and Tables**, **Available Fields**, **Selected Fields**. Selecția poate fi făcută dintr-o tabelă sau o viziune.

pasul 2 – gruparea înregistrărilor: se poate folosi gruparea datelor pentru a împărți pe categorii şi a sorta înregistrările, pentru a fi mai uşor de citit (de exemplu situația încasării impozitelor pe familie, pe sectoare şi centralizat pe municipiu). După alegerea unui câmp în căsuța **Group by**, se pot alege opțiunile **Grouping Options** şi **Summary Option** pentru a îmbunătății selecția. Alegând **Grouping Options** se va deschide caseta de dialog **Grouping Intervals** care permite selectarea nivelurilor de filtrare relevante pentru tipul de date conținute în câmpurile selectate pentru grupare. Alegând **Summary Options** se deschide o cutie de dialog în care utilizatorul poate specifica tipul de calcul care se va aplica unui câmp numeric:

Tipuri de calcul	**Rezultat**
SUM	Suma valorilor din câmpul numeric specificat
AVG	Media valorilor din câmpul numeric specificat
COUNT	Numărarea înregistrărilor care conțin valori non-NULL din câmpul specificat
MIN	Valoarea minimă în câmpul numeric specificat
MAX	Valoarea maximă în câmpul numeric specificat

Se pot alege de asemenea opțiunile **Summary only** sau **No Totals** pentru raport. Câmpurile care sunt selectate pentru grupare nu vor fi disponibile la pasul 3.

pasul 3 – alegerea stilului pentru raport : sunt disponibile următoarele stiluri: **Executive**, **Ledger**, **Presentation**, **Banded**, **Casual**, asistentul prezentând într-un grafic, sub o lupă, un exemplu al stilului ales.

pasul 4 – definirea formatului raportului: când se specifică un număr de coloane sau se selectează una din opțiunile pentru format, asistentul prezintă într-un grafic, sub o lupă, un exemplu.

Dacă s-a făcut opțiunea de grupare la pasul 2, opțiunile **Columns** şi **Field Layout** nu mai sunt disponibile.

pasul 5 – sortarea înregistrărilor: se pot selecta unul sau mai multe câmpuri sau o etichetă de index (TAG) în ordinea în care se doreşte să se sorteze înregistrările (ascendent, descendent).

pasul 6 – terminare (butonul **Finish**): dacă numărul câmpurilor selectate nu încap pe o singură linie pe lățimea paginii de raport, acest câmp va fi înglobat pe linia următoare. Dacă nu vrem să fie înglobat câmpul pe linia următoare, se deselectează opțiunea **Wrap fields that do not fit**. Cu butonul **Preview** se poate vizualiza raportul fără a părăsi asistentul. După salvarea raportului, acesta poate fi deschis ulterior şi modificat cu aplicația **Report Designer**.

6.4. Label Wizard

Acesta este asistentul pentru generarea etichetelor cu date preluate dintr-o tabelă. Pentru lansarea asistentului se alege din meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Label** şi se urmăresc paşii:

pasul 1 – selectarea tabelelor: se selectează o tabelă sau o viziune.

pasul 2 – alegerea tipului de etichetă: asistentul oferă o listă cu tipurile de etichete standard care sunt instalate odată cu VFP. Asistentul afişează de asemenea lista cu tipurile de etichete care au fost create de utilizator, prin folosirea aplicației **AddLabel**.

pasul 3 - definirea formei etichetei: se adaugă câmpurile în ordinea în care trebuie să apară pe etichetă. Se pot folosi semne de punctuație, spații, pentru a formata eticheta. Se foloseşte căsuța **Text** pentru a adăga textul. Pe măsură ce se adaugă elemente în etichetă, asistentul va arăta modificările făcute. Utilizatorul poate verifica imaginea etichetei pentru a vedea dacă câmpurile introduse se încadrează în dimensiuni. De asemenea, se poate folosi butonul **Font** pentru a schimba stilul şi mărimea literelor folosite în etichetă.

pasul 4 – sortarea înregistrărilor: se selectează câmpurile sau etichetele de index (TAG) în ordinea în care doreşte să sorteze înregistrările.

pasul 5 – terminare (butonul **Finish**): utilizatorul poate activa opțiunea **Preview** pentru a verifica restricțiile impuse, înainte de terminare. După salvarea etichetei, aceasta poate fi deschisă şi modificată cu aplicația **Label Designer**.

6.5. Query Wizard

Asistentul de interogare, care pe baza tabelelor sau viziunilor selectate pentru a furniza informații, extrage înregistrările care îndeplinesc criteriile de filtrare şi direcționează rezultatele către tipul de ieşire specificat: fereastră **Browse**, raport, tabelă, etichetă etc. Pentru lansarea asistentului se alege din meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Query Wizard** şi se urmăresc paşii:

pasul 1 – selectarea câmpurilor: se pot selecta câmpuri din diferite tabele sau viziuni. Mai întâi se selectează câmpurile dintr-o tabelă sau viziune, se mută în căsuța Selected fields, utilizând butoanele de adăugare; apoi se selectează câmpurile din altă tabelă sau viziune etc.

pasul 2 – stabilirea relațiilor între tabele: utilizatorul poate selecta cîmpurile comune mai multor fişiere din lista afişată, pentru a stabili relațiile între tabele sau viziuni.

pasul 3 – includerea înregistrărilor: se poate limita interogarea (**Query**) alegând din două tabele, numai liniile care coincid, sau toate liniile dintr-unul din tabele. Implicit numai înregistrările care coincid sunt incluse. Dacă se doreşte crearea unei joncțiuni interne se selectează opțiunea **Only matching row** (numai înregistrările care coincid). Dacă se doreşte crearea unei joncțiuni externe, se selectează toate înregistrările din unul din cele două tabele.

pasul 4 – filtrarea înregistrărilor: se poate reduce numărul înregistrărilor selectate în interogare, prin crearea de expresii logice care să filtreze înregistrările din tabelele sau viziunile alese. Se pot crea două expresii logice, legate cu operatorul **And**, care va permite selectarea numai a acelor înregistrări care satisfac ambele criterii. Utilizarea operatorului **Or** face ca în selecția înregistrărilor, să intre acelea care satisfac doar unul din criterii. Opțiunea **Preview** permite vizualizarea înregistrărilor care vor fi incluse în interogare (**Query**), pe baza aplicării criteriilor de filtrare.

pasul 5 – sortarea înregistrărilor: se pot selecta până la 3 câmpuri (chei de sortare) sau o etichetă de index (TAG) pentru a stabili ordinea în care rezultatele interogării vor fi sortate (ascendent sau descendent).

pasul 6 – limitarea înregistrărilor: se poate limita suplimentar numărul de înregistrări în interogare, fie indicând un procent de vizualizare din numărul de înregistrări cu valorile cele mai mari (sortare descendentă) / mai mici (sortare ascendentă), fie indicând numărul de înregistrări cu valorile cele mai mari/mai mici, care vor fi afişate. De exemplu, pentru a vedea doar primele 10 înregistrări cu valoarea cea mai mare din interogare (sortare descendentă), se introduce la opțiunea **Number of records** valoarea 10 în căsuța **Portion value**. Cu opțiunea **Preview** se pot vizualiza rezultatele interogării pe baza restricțiilor impuse.

pasul 7 – terminare (butonul **Finish**): după ce interogarea a fost salvată, ea poate fi deschisă ulterior, lansând aplicația **Query Designer**.

6.6. Mail Merge Wizard

Acesta este asistentul pentru documente interclasate (fuziune / unire) de tip Word sau fişiere de tip text care sunt compatibile cu orice procesor de text. Pentru a executa aplicaţia, trebuie să fie instalată o versiune Microsoft Word curentă pecum şi protocolul standard pentru servere baze de date, ODBC (Open Database Connectivty). ODBC se instalează pentru diferite tipuri de baze de date şi va permite programului VFP 6.0 să se conecteze la baza de date şi să acceseze datele. Pentru lansarea asistentului se alege din meniul sistem **Tools**, submeniul **Wizard**, opţiunea **Mail Merge** şi se continuă cu paşii:

pasul 1 – selectarea câmpurilor: se pot selecta câmpuri doar dintr-o singură tabelă sau viziune.

pasul 2 – alegerea procesorului de text: trebuie să fie instalată o versiune curentă de Microsoft Word în cazul în care se alege opţiunea Microsoft Word. Dacă se selectează opţiunea **Comma-delimited text file**, asistentul va sări peste paşii 3 şi 4 şi va merge direct în ecranul de terminare.

pasul 3 – selectarea tipului de document: se crează un nou document sau se adaugă date la un document existent.

pasul 4 – selectarea stiluilui documentului: se selectează unul din tipurile de fişiere pe care le recunoaşte editorul Word. Dacă se crează un document nou, se va specifica tipul documentului **Form Letter**, **Label**, **Envelope**, **Catalog**.

pasul 5 – terminare (butonul **Finish**): dacă s-a selectat crearea unui document nou în Word, asistentul va deschide documentul în Microsoft Word. Dacă s-a selectat opţiunea de creare a unui fişier de tip text, asistentul îl va salva.

Introducerea unui câmp al bazei de date în documentul Word se face astfel:

- se poziţionează cursorul în locul din document unde se doreşte inserarea câmpului;

pe bara de instrumente pentru interclasare din opţiunea pop-up **Insert Merge Field** se alege numele câmpului ce trebuie inserat;

- activând opţiunea - **View Merged Data** (vizualizarea datelor care au fuzionat), se poate vedea efectiv valoarea câmpului selectat pentru inserare. Butoanele pentru deplasare la începutul / sfârşitul fişierului sau secvenţial înainte / înapoi, vor muta pointerul de fişier pe înregistrarea corespunzătoare, afişând noua valoare a câmpului, în documentul Word.

6.7. Editorul de texte în VFP 6.0

Mediul VFP 6.0 pune la dispoziția utilizatorului un editor de texte, pentru crearea fişierelor de tip text, aplicație care se deschide într-o fereastră de tip Windows.

6.7.1. Lansarea editorului

Lansarea editorului se face fie cu una din comenzile:
MODIFY FILE <*nume_fişier*> && modificare fişier de tip text;
MODIFY COMMAND <*nume_program*> && modificare program sursă.

introduse în fereastra de comenzi, fie din meniul sistem se alege meniul **File**, submeniul **New**, opțiunea **Program** / **Text File**. În editor se pot folosi pentru deplasare: tastele cu săgeți direcționale, **[Page Up]** - o pagină ecran în sus, **[Page Down]** - o pagină ecran în jos, **[Home]** - la începutul rândului, **[End]** - la sfîrşitul rândului curent.

6.7.2. Lucrul cu blocul de linii din fişier

Asupra unei linii sau unui bloc de linii de text se pot executa următoarele operații:

marcarea blocului – se ține tasta **[Shift]** apăsată şi cu una din săgețile direcționale, sau tastele **[Home]**, **[End]**, [**Page Up**], **[Page Down]** se marchează liniile de text (care vor apare în video-invers).

copierea blocului de linii marcat în Clipboard – se face:

folosind opțiunea **Cut** din meniul **Edit** sau apăsând simultan tastele **[Ctrl]** + **[X]**, textul selectat va fi şters din fişier;

folosind combinația de taste [**Ctrl]**+**[C]**, caz în care textul rămâne în fişier.

inserarea la o nouă poziție în fişier a textului din Clipboard – se face utilizând opțiunea **Paste** din fereastra **Edit** sau combinația de taste **[Ctrl]** + **[V]**.

deselectarea unui bloc de linii – se face fie apăsând una din săgețile direcționale sau butonul stâng de la mouse.

Programele în VFP au ca regulă generală, scrierea instrucțiunilor pe o singură linie (255 caractere). Dacă totuşi o instrucțiune trebuie sa fie scrisă pe mai multe rânduri, la sfârşitul fiecărei linii se pune un caracter de continuare **;** .

7. Elemente ale limbajului propriu Visual FoxPro 6.0

7.1. Simboluri

Pentru scrierea comenzilor (instrucțiunilor) se folosesc:

– literele alfabetului latin, litere mari / litere mici, compilatorul nefăcând diferență între cele două tipuri;

– cifrele 0..9;

– caractere speciale: +, -, *, /, (,), <, >, =, &, @, !, ‘, “, #, $, %, ^, \, |, _, spațiu, ;, virgula, ?.

7.2. Variabile de sistem

Sunt variabile proprii sistemului, predefinite, la care utilizatorul are acces. Folosirea lor în program nu implică operații de inițializare, incrementare. De exemplu, variabila _PAGENO=*n* conține numărul paginii *n* dintr-un raport, cu ajutorul ei putându-se controla afişarea paginii *n* la monitor sau la imprimantă.

7.3. Comentariul

Se foloseşte pentru a da lămuriri asupra a ceea ce face programul sursă, explicații asura acțiunilor programului, numelor variabilelor etc, pentru a indicaînceputul unei linii neexecutabile în program. Sunt admise pentru comentariere:

caracterul * plasat la începutul liniei va comentaria întreagul rând (fără spațiu de demarcare);

caracterele && plasate la începutul / sfârşitul liniei, cu cel puțin un spațiu de demarcare față de textul propriu-zis;

cuvântul rezervat NOTE, pentru a comentaria mai multe rânduri de text, se introduce în continuare prima linie de comentariu (la distanță de minim un spațiu), se marchează continuarea comentariului pe rândul următor cu caracterul punct şi virgulă (;), închiderea comentariului se face cu caracterul punct.

7.4. Tipuri de date, operații şi funcții

Datele (variabile sau constante) utilizate în programare pot fi simple sau sub formă de masiv de date (vectori, matrici, care presupun o ordonare a datelor după un criteriu de poziție). La rândul lor aceste două categorii pot fi alcătuite din date de tip:

- *logic*, care nu pot lua decât două valori: **.T. | .Y.** (adevărat) şi **.F.** | **.N.**(fals);
- *numeric,* numere întregi sau fracționare*;*
- *şir de caractere,* reprezintă o mulțime ordonată de caractere, fiecare caracter având un număr de ordine în şir, numerotarea începând cu 1. Şirurile de caractere (constante) se scriu între apostrof sau ghilimele;
- *calendaristic*, de exemplu 01/12/2007.

Se pot construi expresii, combinând datele (operanzi) cu operații specifice (operatori). Operatorii la rândul lor sunt de tip:

- *logic,* lucrează cu toate tipurile de date şi returnează o valoare logică **(.T.** / **.F.**)

Operator	**Acțiune**	**Cod**
( )	gruparea expresiilor logice	*Cvar*1 AND (*Cvar*2 AND *Cvar*3)
NOT, !	negare logică	IF NOT *Cvar*A = *Cvar*B sau IF ! *Nvar*1 = *Nvar*2
AND	ŞI logic	*LvarX* AND *LvarY*
OR	SAU logic	*LvarX* OR *LvarY*

unde *Cvar* – variabilă de tip caracter, *Nvar* – variabilă de tip numeric, *Lvar* – variabilă de tip logic.

- *relațional,* lucrează cu toate tipurile de date, expresia este evaluată şi se returnează o valoare logică **.T.** / **.F.**(adevărat / fals).

Operator	**Acțiune**	**Cod**
<	mai mic decât	? 23 < 54
>	mai mare decât	? 1 > 2 sau ?.t.>.f.
=	egal cu	? *cVar*1 = *cVar*
<>, #, !=	diferit de	? .T. <> .F.

<=	mai mic egal cu	? {^1998/02/16} <= {^1998/02/16}
>=	mai mare egal cu	? 32 >= *n_ani*
==	comparare şiruri de caractere	? status == "Open"

În cazul variabilelor de tip caracter operatorii relaţionali funcţionează prin compararea codurilor ASCII corespunzătoare caracterelor din cele două şiruri, comparaţia făcându-se caracter cu caracter, de la stânga la dreapta.

Dacă şirurile care se compară nu au aceeaşi lungime, implicit se completează şirul mai scurt cu codul ASCII pentru caracterul 0.

Operaţia de egalitate a două şiruri de caractere este controlată de comanda:

SET EXACT ON|OFF

prin clauza OFF, implicită, se consideră că cele două şiruri sunt egale, în cazul când caracterele coincid pe lungimea celui mai scurt.

prin clauza ON, egalitatea se verifică pe lungimea şirului mai lung, spaţiile de la sfârşitul şirurilor sunt ignorate.

- *numeric,* lucrează cu toate tipurile de date numerice.
-

Operator	**Acţiune**	**Cod**
( )	gruparea subexpresiilor	(4-3) * (12/*Nvar*2)
**, ^	exponent (putere)	? 3 ** 2 sau ? 3 ^ 2
*, /	înmulţire, împărţire	? 2 * 7 sau ? 14 / 7
%	restul împărţirii (modulo)	? 15 % 4
+, -	adunare, scădere	? 4 + 15

Ordinea operatorilor din tabelă este cea folosită în matematică.

Operația modulo (%) se mai poate executa apelând funcția MOD(n1,n2) care va calcula restul împărțirii numărului n1 la numărul n2.

- *caracter,* care permite concatenarea, compararea datelor de tip şir de caracatere, utilizând următorii operatori:

Operator	Acțiune	Cod
+	*Concatenare*. Se unesc două şiruri de caractere, un şir şi un câmp dintr-o înregistrare sau un şir de caractere şi o variabilă	? 'Bună ' + 'ziua'
-	*Concatenare.* Se îndepărtează spațiile goale de la sfârşitul elementului care precede operatorul, apoi se unesc cele două elemente.	? 'Bună ' + 'seara'
$	*Comparare.* Se caută un şir de caractere în interiorul altui şir de caractere.	? 'punct' $ ‘contrapunct' ? 'principal' $ clienti.adresa

7.4.1. Date de tip numeric. Funcții aritmetice şi financiare

Expresiile de tip numeric pot fi:

- câmpuri de tip numeric (N), întreg (I), virgulă mobilă (F) dintr-o tabelă;
- funcții care returnează o variabilă (constantă) numerică;
- variabile şi constante de tip numeric.

Scop / reprezentare matematică	Funcție VFP
restul împărțirii exacte a lui n_1 la n_2	mod (n_1,n_2)
$\lvert x \rvert$	abs(x)
partea întreagă a lui x	int(x)
următorul întreg mai mare sau egal cu x (plafon)	ceiling(x)
următorul întreg care este mai mic sau egal cu x (podea)	floor(x)
rotunjirea unei expresii numerice la un număr specificat de zecimale	round(*nExpr*,*nr_zecimale*)
e^x	exp(x)
ln(x)	log(x)

$\log_{10} x$	log10(*x*)
$\sqrt{x}$	sqrt(*x*)
sin *x*	sin(*x*)
cos *x*	cos(*x*)
tg *x*	tan(*x*)
arcsin *x*	asin(*x*)
arccos *x*	acos(*x*)
arctg *x*	atan(*x*)
trecerea din grade în radiani	dtor(*x*)
trecerea din radiani în grade	rtod(*x*)
alegerea numărului de zecimale *n*	set decimals to *n*
valoarea constantei π	PI()

Funcții financiare

Scop	Funcție VFP
returnează valoarea ratei necesare pentru achitarea unui împrumut *s*, cu dobânda *d**, pe perioada *nr*** (annual sau lunar)	PAYMENT(*s,d,nr*)
returnează suma ce trebuie depusă în cont pentru a plăti o rată *s*, pe o perioadă *nr* (ani sau luni), dacă dobânda acordată este *d*	PV(*s,d,nr*) (Present Value)
returnează suma ce se poate strânge în cont, după o perioadă *nr* (ani sau luni), dacă dobânda este *d* iar rata depunerii este *s*	FV(*s,d,nr*) (Future Value)

*Argumentul *d* (dobânda) este utilizat în formă zecimală (*d*=0.06 | 6%). Dacă dobânda se preia sub formă procentuală se va împărții la 100. Implicit este dobânda anuală.

**Argumentul *nr* (perioada) este implicit exprimat în ani. Dacă calculul se efectuează pentru o perioadă exprimată în luni, va terbui modificată corespunzător şi dobânda (anuală | lunară).

7.4.2. Aplicații ale funcțiilor financiare

A. Funcția PAYMENT returnează valoarea fiecărei plăți, dintr-o serie periodică de plăți (rată), a unui împrumut *s*, cu dobînda fixă *d,* pe o perioadă *nr*.

Sintaxa funcției: PAYMENT(*s,d,nr*)

Exemplul 1. Firma DAEWOO vinde autoturisme marca MATIZ care pot fi achitate în rate lunare, pe o perioadă de trei ani. Împrumutând de la BRD suma de 6000 $ cu dobânda anuală de 5.3%, să se stabilească care este rata lunară pe care trebuie să o achite un client.

În fereastra de comenzi introducem comanda pentru crearea fişierul sursă IMPR:

MODI COMM IMPR
se lansează editorul de programe VFP şi se introduc liniile de comenzi (instrucțiuni):
SET TALK OFF && anulează afişarea rezultatelor comenzilor
CLEAR && curăță spațiul de afişare
SET CURRENCY TO '$' && definirea simbolului monedei
STORE 0 TO S,NR && inițializarea variabilelor S, NR cu 0
D=0.00 && inițializarea variabilei D
@5,20 SAY 'STABILIREA RATEI LUNARE PENTRU UN IMPRUMUT'
@7,15 SAY 'SUMA IMPRUMUTATA ?:' GET S
NOTE afişarea la coordonate fixate de carcaterul @ (linia 7, coloana 15) a mesajului dintre apostrof şi editarea variabilei S.
@9,15 SAY 'DOBANDA ANUALA (%) ?:' GET D
@11,15 SAY 'IN CATE LUNI TREBUIE ACHITAT IMPRUMUTUL ?:' GET NR
READ && citirea întregului ecran – variabilele editate cu @.. GET
D=D/12/100 && transformare dobândă anuală procentuală, în dobândă lunară, exprimată zecimal
@15,20 SAY 'RATA LUNARA ESTE DE '
SET CURRENCY RIGHT && setarea afişării simbolului monedei la dreapta
@15,45 SAY PAYMENT(S,D,NR) FUNCTION '$9999.99' && afişarea rezultatului funcției şi a monedei
WAIT WINDOW 'APASATI ENTER' && păstrează ecranul cu rezultate până se apasă Enter

Fereastra de editare se închide cu **[CTRL]** + **[W]** sau din butonul de închidere şi se salvează fişierul. Lansarea în execuție a programului se face fie din meniul sistem **Program**/**Do**, fie din bara de instrumente se activează icon-ul cu semnul exclamării, sau în fereastra de comenzi se introduce comanda
DO IMPR
Rata lunară este 180.63 $.

B. Funcția PV returnează valoarea sumei care trebuie să existe în cont, pentru a putea plăti o rată *s*, pe o perioadă *nr*, pentru o dobândă anuală *d* acordată de bancă la depozit.

Sintaxa funcției: PV(*s,d,nr*)

Exemplul 2. O firmă trebuie să achite o chirie lunară de 300 $ pentru sediu, pe o perioadă de 5 ani. Câți bani trebuie depuşi în cont la banca BCR, dacă dobânda la acest tip de depozit este de 4% pe an.

În fereastra de comenzi se introduce comanda pentru crearea fişierului CONT:

MODI COMM CONT
se lansează editorul de programe VFP şi se introduc liniile de comenzi:

```
SET CLOCK ON && afişarea cesul cu ora dată de sistemul de operare
CLEAR
STORE 0 TO S,NR
D=0.00
@5,25 SAY 'STABILIREA SUMEI NECESARE IN CONT PENTRU PLATA UNEI RATE
LUNARE'
@7,20 SAY 'RATA LUNARA ?:' GET S
@9,20 SAY 'DOBANDA ANUALA (%) ?:' GET D
@11,20 SAY 'PERIOADA (LUNI) DE ACHITARE A RATEI ?:' GET NR
READ
D=D/12/100
@13,25 SAY 'SUMA INITIALA DIN CONT TREBUIE SA FIE DE' ;
+ALLTRIM(STR(PV(S,D,NR),10,2))+' $'
NOTE funcția STR() transformă o valoare numerică într-un şir de caractere;
funcția ALLTRIM() elimină spațiile de la începutul şi sfârşitul şirului.
WAIT WINDOW 'APASATI ENTER'
```

Se salvează programul şi se lansează în execuție cu comanda:

```
DO CONT
```

Suma necesară este 16289.72$.

C. Funcția FV returnează valoarea sumei ce se strânge într-un cont, în cazul în care se depune o rată *s*, banca oferă o dobândă anuală *d*, pe o perioadă *nr*.

Sintaxa funcției: FV(*s,d,nr*)

Exemplu. Ce sumă va strânge în cont un copil, care primeşte pensie alimentară timp de 10 ani, câte 50 $ pe lună, la o bancă ce oferă o dobândă de 4% pe an ?

În fereastra de comenzi se introduce comanda pentru crearea fişierului SUMA:

MODI COMM SUMA

se lansează editorul de programe VFP şi se introduc liniile de comenzi:

```
CLEAR
STORE 0 TO S,NR
D=0.00
@5,25 SAY 'STABILIREA SUMEI STRANSE  IN CONT DUPA O PERIOADA DE TIMP'
@7,20 SAY 'RATA LUNARA ?:' GET S
@9,20 SAY 'DOBANDA ANUALA ?:' GET D
@11,20 SAY 'PERIOADA (LUNI) DE DEPUNERE A RATEI ?:' GET NR
READ
D=D/12/100
@13,25 SAY 'SUMA STRANSA IN CONT ESTE '+ALLTRIM(STR(FV(S,D,NR),10,2))+' $'
WAIT WINDOW 'APASATI ENTER'
```

Se salvează programul şi se lansează în execuție cu comanda:

DO SUMA

Suma strânsă în cont este 7462.49$.

7.4.3. Date şi funcții de tip caracter

Operanzii de tip caracter pot fi:

- câmpurile de tip CHARACTER ale unei tabele;
- funcțiile care returnează un şir de caractere;
- variabile şi constante de tip şir de caractere.

Dacă şirul de caractere conține în componența sa caracterul ‘ (“) atunci pentru definirea şirului se folosesc caracterele “ (‘).

Scop	Funcție VFP
Returnează caracterul corespunzător codului ASCII *n* (I=1...127)	CHR(*n*)
Returnează codul ASCII pentru un caracter *c*	ASC(‘*c*’)
Extragerea unui subşir de de caractere, de lungime n_2, din şirul *s*, începând cu poziția n_1	SUBSTR(*s*,*n*1,*n*2)
Extragerea unui subşir de *n* caractere, începând din stânga şirului *s*	LEFT(*s*,*n*)
Extragerea unui subşir de *n* caractere, începând din dreapta şirului *s*	RIGHT(*s*,*n*)
Returnarea unui şir de caractere *s,* în mod repetat de *n* ori	REPLICATE(*s*,*n*)
Obținerea unui şir de *n* spații goale	SPACE(*n*)
Eliminarea spațiilor într-un şir de caractere:	
de la începutul şi sfârşitul şirului *s*	ALLTRIM(*s*)
de la începutul şirului *s*	LTRIM(*s*)
de la sfârşitul şirului *s*	RTRIM(*s*)
Adăugarea de spații, sau a unui caracter c_pad, într-un şir *s* pentru a ajunge la lungimea *n*:	
la ambele capete	PADC(*s*,*n*[, *c_pad*])
la stânga	PADL(*s*,*n*[,*c_pad*])
la dreapta	PADR(s,n[,c_pad])
Returnează lungimea unui şir de caractere *s*	LEN(*s*)
Returnează poziția de început, la a *n*-a apariție, a unui subşir de caractere *ss* într-un şir de caractere *s*	AT(s,ss,*n*)
Returnează poziția de început, la a *n*-a apariție, a unui subşir de caractere *ss* într-un şir de caractere *s*, fără a se ține seama de litere mari/mici	ATC(*s*,*ss*,*n*)
Returnează poziția numerică, la a *n*-a apariție, a unui subşir de caractere *ss* într-un şir de caractere *s*, începând căutarea de la dreapta	RAT(*s*,*ss*,*n*)
Transformarea caracterelor unui şir s în litere mari	UPPER(*s*)
Transformarea caracterelor unui şir *s* în litere mici	LOWER(*s*)
Transformarea primului caracter al unui şir *s*, dacă este o literă, în majusculă	PROPER(*s*)

7.4.4. Date şi funcții de tip calendaristic

Expresiile de tip dată calendaristică pot fi:

- câmpuri de tip dată calendaristică (DATE), dintr-o tabelă;
- funcții care returnează data calendaristică;
- constante de tip dată calendaristică.

O dată de tip dată calendaristică precizează ziua, luna, anul, ordinea acestor 3 elemente poate fi aleasă din 11 moduri, cu ajutorul comenzii:

SET DATE TO <tip_dată>

unde:

tip_dată	Format
AMERICAN	ll/zz/aa
ANSI	aa.ll.zz
BRITISH	zz/ll/aa
FRENCH	zz/ll/aa
GERMAN	zz.ll.aa
ITALIAN	zz-ll-aa
JAPAN	aa/ll/zz
USA	ll-zz-aa
MDY	ll/zz/aa
DMY	zz/ll/aa
YMD	aa/ll/dd

Cele 3 elemente pot fi separate cu '/', '-', '.'. Implicit se consideră formatul: *ll*/*zz*/*aa*.

Cu comanda:

SET CENTURY ON|OFF

se poate preciza dacă anul este este afişat cu 2 cifre (ON) sau 4 cifre (OFF).

Comanda:

SET MARK TO <*caracter*>

precizează ce caracter se va folosi ca separator între cele 3 elemente ale datei.

Funcții pentru datele de tip dată calendaristică sunt date în tabelul următor:

Scop	Funcție VFP
Returnează data din sistem(calculator)	DATE()
Numele zilei dintr-o expresie de tip dată *d*	CDOW(*d*)
A câta zi din săptămână dintr-o expresie de tip dată *d*	DOW(*d*)
Numele lunii dintr-o expresie de tip dată *d*	CMONTH(*d*)
A câta lună din an dintr-o expresie de tip dată *d*	MONTH(*d*)
Izolarea anului dintr-o expresie de tip dată *d*	YEAR(*d*)
Returnează ora din sistem (calculator)	TIME()
Returnarea sub formă de şir de caractere, în format *aaaallzz* a unei expresii de tip dată calendaristică *d*	DTOS(*d*)

Folosirea într-un program a unei constante de tip dată calendaristică, se face între acolade ({^31/01/01}).

7.4.5. Date de tip memo

Acest tip de date este asemănător tipului şir de caractere. Este indicată folosirea datelor de tip memo în cazul în care un câmp al unei înregistrări dintr-o tabelă, nu are o lungime cunoscută (sau care nu poate fi aproximată). Utilizarea unui câmp memo într-o tabelă, are ca efect asocierea la tabelă a unui fişier suplimentar în care se depun datele câmpului memo. În tabelă, în câmpul memo sunt stocate informații referitoare la tabela suplimentară. Lungimea unui câmp memo este 10 octeți.

7.5. Variabile şi masive

7.5.1. Variabile

O variabilă are asociate următoarele elemente:

- numele
- conținutul (valoarea)
- tipul variabilei
- zonă de memorie

Din punct de vedere al utilizării lor, variabilele pot fi:

- *locale*, acționează într-o funcție sau procedură. Declararea utilizării lor se face cu comanda:

LOCAL <*listă_variabile*>

<*listă_variabile*> - variabilele se declară prin nume şi sunt separate în listă prin virgulă.

- *globale*, acționează la nivelul întregului program, inclusiv în funcții şi proceduri. Declararea utilizării lor se face cu comanda:

PUBLIC <*listă_variabile*>

Atribuirea de valori unei variabile se face cu sintaxa:

<*nume_var*>=<*expresie*>

Efectul comenzii: se evaluaeză expresia din dreapta, se caută dacă variabila a fost definită şi i se atribuie valoarea şi tipul expresiei.

Atribuirea se mai poate face utilizând comanda:

STORE <*expresie*> TO <*listă_variabile*>

Efectul comenzii: se evaluază expresia şi se atribuie variabilelor din listă, valoarea şi tipul ei.

Eliberarea zonelor de memorie ocupate de variabile care nu mai sunt necesare în program se face cu una din comenzile:

RELEASE [ALL] <*listă_variabile*>

CLEAR [ALL] <*listă_variabile*>

Afişarea conținutului variabilelor existente în memorie se face cu comenda:

DISPLAY MEMORY [TO PRINTER[PROMPT] | TO FILE *nume_fişier*] [NOCONSOLE]

unde:

TO PRINTER[PROMPT] – ieşirea la imprimantă. Clauza PROMPT se foloseşte pentru a confirma imprimarea, într-o fereastră de dialog;

TO FILE *nume_fişier* – ieşirea direcționată către un fişier;

[NOCONSOLE] – împiedică afişarea în fereastra principală VFP, a rezultatelor comenzii.

7.5.2. Macrosubstituția

Macrosubstituția tratează conținutul unei variabile ca un şir de caractere în sens literal (câmpul unei înregistrări dintr-o tabelă, nume de fişier etc.), ca şi cum în locul variabilei respective ar fi pus şirul de caractere fără apostrofuri.

Sintaxa comenzii:

& *nume_variabilă*

Exemplu. Presupunem că există 2 fişiere ESTUD1.DBF şi ESTUD2.DBF. Se doreşte să li se vizualizeze conținutul într-un ciclu FOR.

PUBLIC FIS C(20)

```
FOR I=1 TO 2
    FIS='ESTUD'+ALLTRIM(STR(I))+'.DBF'
    USE &FIS
    BROWSE
ENDFOR
CLOSE DATABASES
```

7.5.3. Masive de date

Masivele de date (vectori şi matrice) pot fi privite ca variabile sau constante, care au o structură compusă, identificarea în cadrul structurii făcându-se pe baza unui criteriu de poziție. Pentru vector criteriul de poziție este numărul de ordine al elementului, pentru matrice, criteriul de poziție este fie numărul liniei şi al coloanei, fie liniarizând matricea, se poate adresa un element prin poziția care o ocupă în masiv (de exemplu, într-o matrice *A* de dimensiune 3x3, elementul *A*(3,2) este al 8-lea element din matrice şi se poate adresa cu *A(*8)). Prin liniarizare o matrice poate fi adresată ca un vector. Calculul poziției în matricea liniarizată (vector), pe baza coordonatelor linie/coloană se poate face cu formula:

poziție=(*linie*-1)**Nr_linii*+*coloană*

Utilizarea masivelor de date comportă următoarele aspecte:

- atribuirea unui nume pentru identificare;
- declararea tipului de masiv: vector sau matrice (rezervare zonă de memorie);

Masivele de date pot fi declarate în două moduri:

- fie la începutul programului sau al subprogramului, utilizând comanda PUBLIC (*variabiă globală*) sau LOCAL (*variabilă locală*):

PUBLIC [ARRAY] *nume_masiv*1(*n_linie*[,*n_col*])[, *nume_masiv*2(*n_linie*[,*n_col*])...
LOCAL [ARRAY] *nume_masiv*1(*n_linie*[,*n_col*])[, *nume_masiv*2(*n_linie*[,*n_col*])...

Variabilele create cu PUBLIC / LOCAL sunt inițializate cu valoarea fals (**.F.**). Variabilele locale sunt legate de funcția sau procedura în care au fost declarate, dar pot fi transmise prin referință;

- fie în interiorul programului utilizând una din comenzile:

DIMENSION [ARRAY] *nume_masiv*1(*n_linie*[,*n_col*])[, *nume_masiv*2(*n_linie*[,*n_col*])...
DECLARE [ARRAY] *nume_masiv*1(*n_linie*[,*n_col*])[, *nume_masiv*2(*n_linie*[,*n_col*])...

Elementele masivului sunt inițializate cu valoarea fals ca şi în primul caz.

După declarare, masivul poate fi inițializat global (toate elementele capătă aceeaşi valoare) şi cu această operație se stabileşte tipul de date al elementelor masivului:

STORE <*val_init*> TO <*nume_masiv*>

Exemplu

DIMENSION MAT(5,7)
STORE 0 TO MAT

Tipul şi dimensiunile unui masiv de date pot fi modificate în cadrul aceluiaşi program, prin noi declarații DIMENSION, DECLARE.

7.6. Funcții de prelucrare a masivelor

Funcția ALEN (*nume_masiv,n*)

Returnează următoarele informații:

- numărul de elemente, dacă *n*=0;
- numărul de linii, dacă *n*=1;
- numărul de coloane, dacă *n*=2.

Funcția AINS(*nume_masiv,n*[,2])

Are ca efect:

- inserarea unui element nou într-un vector, inaintea elementului de pe poziția *n,*dacă argumentul 2 lipseşte;
- inserarea unei linii într-o matrice, înaintea liniei cu numărul *n*, dacă argumentul 2 lipseşte;
- inserarea unei coloane într-o matrice, înaintea coloanei cu numărul *n*, dacă argumentul 2 apare.

Funcția returnează valoarea 1 dacă inserarea s-a efectuat cu succes.

Observație. Inserarea unui element, a unei linii sau coloane într-un masiv nu modifică dimensiunea masivului, ci duce la pierderea elementelor de pe poziția *n*.

Funcția ADEL(*nume_masiv,n*,[2])

Are ca efect operația inversă comenzii AINS, deci ştergerea elementului, liniei, coloanei de pe poziția *n*. Argumentul 2 trebuie să apară în cazul în care ştergem o coloană dintr-o matrice.

Prin cele două funcții, elementele unui masiv sunt translatate cu o poziție la dreapta (AINS) sau la stânga (ADEL). Funcția va returna valoarea 1 dacă ştergerea liniei/coloanei s-a efectuat cu succes.

Funcția ACOPY(*masiv_sursă,masiv_destinație*[,*nr_încep_sursă*[,*nr_elem_copiate*[, *nr_încep_destin*]]])

Este comanda de copiere a elementelor unui masiv (sursă) în elementele altui masiv (destinație), unde:

- *masiv_sursă, masiv_destinație* – numele masivelor sursă/destinație;
- *nr_încep_sursă* – numărul elementului din sursă de la care începe copierea;
- *nr_elem_copiate* – numărul de elemente ce vor fi copiate din masivul sursă, dacă are valoarea –1 toate elementele începând cu poziția *nr_încep_sursă* vor fi copiate;
- *nr_încep_destin* – numărul elementului din masivul destinație de la care începe inserarea.

Funcția returnează numărul elementelor copiate în masivul destinație.

Funcția ASORT(*nume_masiv*[,*nr_încep*[,*nr_elem_sortate*[,*nr_ordine_sortare*]]])

Comanda are ca efect sortarea elementelor unui masiv, putând preciza primul element de la care începe sortarea (*nr_încep*), câte elemente vor fi sortate (*nr_elem_sortate*). Ordinea de sortare (*nr_ordine_sortare*) va fi ascendentă dacă argumentul este 0 sau omis, sau descendentă dacă argumentul este 1 sau orice valoare diferită de 0.

Funcția va returna valoarea 1 dacă sortare s-a efectuat cu succes, sau –1 în caz contrar.

Exemplul 1. Secvența următoare de comenzi va crea o matrice cu 3 linii şi 2 coloane, elementele masivului se identifică prin numărul de ordine în matricea liniarizată:

```
DIMENSION LIT(3,2)
LIT(1) = 'G'
LIT(2) = 'A'
LIT(3) = 'C'
LIT(4) = 'Z'
LIT(5) = 'B'
```

LIT(6) = 'N'

Elementele din matrice vor fi distribuite astfel:

Linia	**Coloana 1**	**Coloana 2**
1	G	A
2	C	Z
3	B	N

Se sorteză masivul cu comanda ASORT(LIT,1)
Sortarea începe cu primul element al masivului (1,1). Elementele din prima coloană a masivului sunt plasate în ordine ascendentă prin rearanjarea liniilor masivului.

Linia	Coloana 1	Coloana 2
1	B	N
2	C	Z
3	G	A

Apoi masivul este sortat începând cu elementul 4 (2,2) din masiv cu comanda ASORT(LIT,4) Elementele din coloana a 2-a sunt plasate în ordine ascendentă prin rearanjarea liniilor masivului, începând cu elementul 4 din masiv.

Linia	Coloana 1	Coloana 2
1	B	N
2	G	A
3	C	Z

Dacă sortarea se efectuează cu succes funcția returnează valoarea 1, în caz contrar valoarea –1.
Pentru a putea fi sortate, elementele masivului trebuie să fie de acelaşi tip.

Exemplul 2.

PUBLIC DIMENSION LIT(3,2)
CLEAR

```
LIT(1) = 'G'
LIT(2) = 'A'
LIT(3) = 'C'
LIT(4) = 'Z'
LIT(5) = 'B'
LIT(6) = 'N'
@1,1 SAY "MATRICEA INITIALA"
K=1
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@6,1 SAY "MATRICEA SORTATA INCEPAND CU PRIMUL ELEMENT"
K=6
ASORT(LIT,1)
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@11,1 SAY "MATRICEA SORTATA INCEPAND CU ELEMENTUL 4"
K=11
ASORT(LIT,4)
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@16,1 SAY "INSERAREA UNEI LINII INAINTE DE LINIA 3"
AINS(LIT,3)
LIT(3,1)="2"
LIT(3,2)="5"
K=16
```

```
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@21,1 SAY "STERGEREA LINIEI 3"
K=21
ADEL(LIT,3)
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
```

În urma execuției programului de mai sus, se obțin rezultatele din Figura 1.

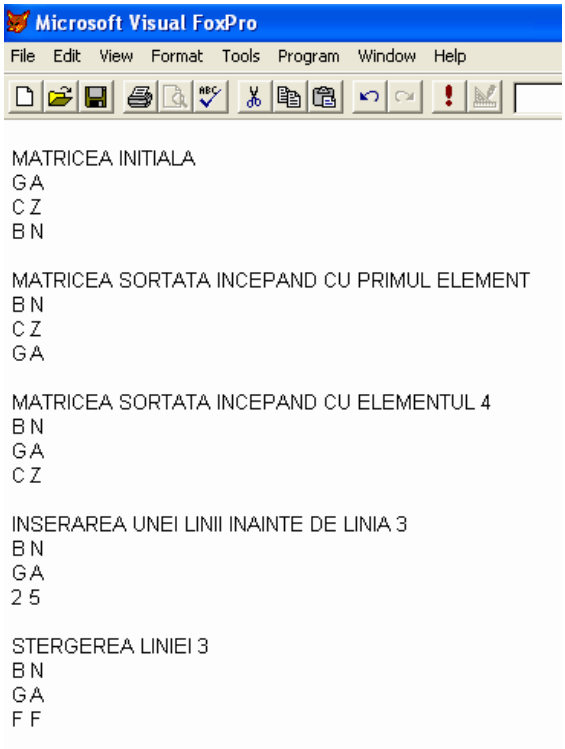


Fig. 1

8. Accesul şi actualizarea datelor

Lucrul cu baze de date şi tabele în cadrul limbajului propriu VFP, presupune trei activităţi, care segmentează construcţia unei aplicaţii:

- deschiderea/crearea bazei de date, a tabelelor, a fişierelor asociate (index);
- exploatarea bazei de date şi a tabelelor, ceea ce implică operaţii de actualizare şi consultare, la nivel de structură sau înregistrări;
- închiderea/ştergerea bazei de date, a tabelelor sau a fişierelor asociate (index).

Comenzile pentru operaţiile de deschidere, creare, închidere a unei baze de date (OPEN DATABASES..., CREATE DATABASES…, CLOSE DATABASES), a unei tabele de date (USE..., CREATE TABLE..., CLOSE TABLES, USE), a unor fişiere de index (USE...INDEX..., INDEX ON..., CLOSE INDEX) în modul de lucru comandă (interpretor), au aceeaşi sintaxă şi în cadrul programelor scrise în limbajul propriu VFP.

8.1. Manipularea structurii unei tabele

Modificarea structurii unei tabele se face cu comanda

MODIFY STRUCTURE

Dacă nu este deschisă nici o tabelă în zona de lucru curentă, se deschide caseta de dialog **Open** care permite utilizatorului selectarea (deschiderea) unei tabele. Comanda are ca efect lansarea aplicaţiei **Table Designer** (proiectantul de tabele) cu ajutorul căruia se fac modificările de structură (adăugare şi ştergere de câmpuri, modificare nume câmp, mărime, tip, adăugarea / ştergerea / modificarea etichetelor de index, specificarea valorilor de tip **NULL**).

Vizualizarea structurii unei tabele se face cu comanda

DISPLAY STRUCTURE [IN <*nr_zonă_de_lucru*> | <*alias_tabelă*>] [TO PRINTER [PROMPT] [TO FILE <*nume_fişier*>] [NOCONSOLE] IN <*nr_zonă_de_lucru*> | <*alias_tabelă*>

Are ca efect afişarea structurii tabelei dintr-o zonă de lucru, alta decât cea curentă, sau a unei tabele adresată prin alias. Semnificaţia clauzelor este următoarea:

- TO PRINTER [PROMPT] – direcţionează informaţiile la imprimantă.

- Clauza PROMPT determină afişarea unei casete de dialog înainte de ieşirea la imprimantă (se pot face setări legate de imprimantă).
- TO FILE <*nume_fişier*> - direcționează ieşirea informațiilor într-un fişier.
- NOCONSOLE – suprimă afişarea datelor în ferestra principală a VFP.

Se mai poate folosi şi comanda LIST STRUCTURE care are aceleaşi clauze.

Copierea structurii unei tabele

Crearea unei noi tabele care va conține structura tabelei active se face cu comanda

COPY STRUCTURE EXTENDED TO <*nume_fişier*> [FIELDS <*listă_câmpuri*>]

Numai câmpurile din lista de câmpuri vor fi incluse în înregistrarea din noul fişier. Dacă clauza FIELDS este omisă, vor fi incluse toate câmpurile tabelei sursă.

8.2. Ordonarea datelor

Datele dintr-o tabelă se pot ordona după un singur criteriu sau după mai multe (ordonare multicriterială) . Ordonarea poate fi ascendentă, dacă fiecare câmp după care se face ordonarea are o valoare mai mare decât câmpul corespunzător din înregistrarea anterioară, sau descendentă. Dacă ordonarea se face multicriterial, se defineşte o cheie primară, care este principalul criteriu de ordonare şi chei secundare. În cazul în care două sau mai multe înregistrări au aceaşi valoare pentru cheia principală, se utilizează cheile secundare pentru departajare, în ordinea în care au fost declarate. În VFP ordonarea unui fişier se poate face în două moduri:

- folosind comanda SORT, prin care se crează un nou fişier cu înregistrările sortate după criteriul ales.
- folosind metoda indexării, care constă în crearea unui fişier de index, în care se stochează ordinea înregistrărilor din tabela sursă.

Comanda SORT are sintaxa:

SORT TO <*nume_tabela*>ON <*nume_câmp*_1> [/A | /D] [/C] [, <*nume_câmp*_2> [/*A* | /*D*] [/*C*] ...] [ASCENDING | DESCENDING] [*Scope*] [FOR <*expresie*_1>] [WHILE <*expresie*_2>] [FIELDS <*lista_câmpurilor*>]

unde:

- *nume_tabela* – numele fişierului sortat.
- *nume_câmp*_1 – specifică câmpul din tabela nesortată asupra căreia se aplică procedura. Implicit sortarea este ascendentă. Nu se pot sorta câmpuri memo sau general. Este cheia primară după care se face sortarea.
- /*A*, /*D*, /*C* – specifică ordinea: *A*: ascendentă, *D*: descendentă, *C*: fără a se ține cont de litere mari/mici.
- *nume_câmp*_2 – specifică un câmp cu rol de cheie secundară.
- ASCENDING – specifică o sortare ascendentă pentru câmpurile care nu sunt urmate de opțiunea /*D*.
- DESCENDING – specifică o sortare descendentă pentru câmpurile care nu sunt urmate de opțiunea /*A*.
- *Scope* – specifică domeniul de sortare (tot fişierul sau doar o parte din înregistrări) **All**, **Next** *n*, **Record** *n*, **Rest**.
- FOR *expresie*_1 – precizează care înregistrări participă la sortare, respectiv acelea pentru care expresie_1 are valoarea **True**
- WHILE *expresie*_2 – precizează condiția pentru care atâta vreme cât au valoarea **True**, înregistrările vor fi sortate.
- FIELDS *lista_câmpurilor* – precizează câmpurile din tabela originală care vor fi incluse în noul fişier sortat. Dacă se omite clauza, toate câmpurile din tabela originală vor fi incluse în tabela sortată.

Exemplu. Să se sorteze fişierul ESTUD1, într-un nou fişier SESTUD1, după câmpul NUME, ascedent, fără să se țină seama de litere mici/mari, luând doar studenții cu media peste 8.00 şi câmpurile MATR, NUME, MBAC.

```
USE ESTUD1
SORT TO SESTUD1 ON NUME/A/C FOR MBAC>8.00 FIELDS MATR,NUME,MBAC
USE SESTUD1
BROWSE
CLOSE DATABASES
DELETE FILE SESTUD1.DBF
```

Sortarea este un procedeu care, în cazul tabelelor mari, duce la mărirea timpului de execuție şi a spațiului solicitat pe hard disk. Se consideră că este mai eficientă folosirea metodei indexării.

Prin metoda indexării se crează un fişier de index, care va conține cel puțin un câmp al tabelei de date, sortat după un criteriu alfabetic, numeric sau cronologic. Fişierul de index este o sortare virtuală a unei tabele, înregistrările din tabela de date rămânând nesortate. Fişierul de index conține informații de localizare a înregistrărilor dintr-o tabelă.

Fişierele index sunt de 2 tipuri:

- *simple* – au extensia IDX, conțin o singură cheie de indexare şi deci un singur criteriu de ordonare.
- *compuse* – au extensia CDX (Compound Index File), în care se stochează mai multe criterii de ordonare, fiecare având asociat un nume (TAG - eticheta de index), corespunzător mai multor chei de indexare.

Crearea unui fişier de index se face cu comanda:

INDEX ON <*expr*> TO <.IDX *file*>|TAG<*tag_name*>[OF<.CDX *file*>] [FOR <*exprL*>]
[ASCENDENT | DESCENDENT] [UNIQUE] [ADDITIVE]

unde:

- <*expr*> - expresia indexului care include câmpuri din tabela ce va fi indexată. Tipurile de expresii admise sunt: *C*, *D*, *N*.
- TO <.IDX *file*> - specifică numele fişierului de index care va fi creat.
- TAG<*tag_name*> OF<.CDX *file*> - permite crearea unui fişier cu mai multe intrări (etichete de index – TAG-uri), numărul lor fiind limitat doar de spațiul pe hard disk.

  Fişierele cu extensia .CDX pot fi:
  - *structurale* – se deschid automat odată cu deschiderea tabelei de date şi au acelaşi nume.
  - *nestructurale* – deschiderea se face indicând un nume la clauza OF<.CDX *file*>, diferit de cel al tabelei de date.
- FOR <*exprL*> - permite ca indexul creat să acționeze ca un filtru asupra tabelei.
- ASCENDENT|DESCENDENT – specifică ordinea de sortare în fişierul de index.
- UNIQUE – pentru valori repetate ale cheii, specifică includerea în fişierul de index numai la prima apariție.
- ADDITIVE – permite crearea unui fişier de index pentru o tabelă, chiar dacă mai sunt deschise alte fişiere de index.

Indexarea după mai multe câmpuri este similară sortării multiple. Nu se pot face indexări multiple pe câmpuri cu tipuri diferite, deci trebuie aduse la o formă comună (şir de caractere).
Comanda

INDEX ON <*câmp_car*> + STR(*câmp_numeric,nc*) + DTOS(*câmp_dată*) TO <*fişier_index*>

permite indexarea după 3 câmpuri diferite.

Semnificaţia parametrilor şi a clauzelor:

- STR(*câmp_numeric,nc*) – transformă un număr într-un şir de caractere,
  nc – numărul maxim de caractere pe care se reprezintă câmpul numeric.
- DTOS(*câmp_data*) – transformă o variabilă de tip dată în şir de caractere.

Pentru deschiderea simultană a tabelei de date şi a fişierelor de index asociate se foloseşte comanda:

USE <*nume_tabelă*> INDEX <*listă_fişiere_index*>

Exemplul 1. Presupunem că este creată o tabelă (STUD) cu următoarea structură:

matricola	N(5)
nume	C(40)
data_n	D
adresa	C(40)

Să se indexeze după câmpul matricola. Tabela are următoarele înregistrări:

nr_înreg	matricola	nume	data_n	adresa
1	215	Aldea I. Dan	22/03/79	Bucureşti
2	110	Barbu A. Vasile	10/05/80	Craiova
3	300	Alexandru F. Ion	13/07/80	Iaşi
4	200	Dinu A. Adrian	25/08/79	Bucureşti
5	210	Cornea C. Ilie	16/07/80	Ploeşti
6	250	Ene V. Dan	24/03/80	Bucureşti

Secvența de comenzi

```
CLEAR
USE STUD
BROWSE
INDEX ON MATRICOLA TO IMATR.IDX
* Obținerea numelui fişierului IDX deschis pentru baza de date curentă sau cea specificată
?? NDX(1)
* Afişează primul fişier index activ
BROWSE
* Înregistrările din fişierul de date apar sortate după câmpul matricola.
CLOSE DATABASES
CLEAR
```

crează fişierul index IMATR.IDX. Acesta are structura:

matricola	**nr_înreg**
110	2
200	4
210	5
215	1
250	6
300	3

Exemplul 2. Să se indexseze tabela STUD după câmpul NUME şi să se creeze un fişier structural STUD.CDX.

```
USE STUD
INDEX ON UPPER(NUME) TAG NUME
BROWSE
CLOSE DATABASES
```

Exemplul 3. Să se indexeze tabela ESTUD1 după câmpul NUME, să se creeze un fişier nestructural INUM.CDX, ordonat ascendent după valorile cheii:

```
USE ESTUD1
INDEX ON UPPER(NUME) TAG NUME OF INUM.CDX ASCENDING
BROWSE
CLOSE DATABASES
```

Exemplul 4. Utilizând tabela ESTUD1 să se indexeze tabela după o cheie compusă (NUME, MBAC), fără a permite existența dublurilor, într-un fişier de index INM.IDX. Structura tabelei este: MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), ANIF N(4), DATAN N(4), MBAC N(5,2).

```
USE ESTUD1
INDEX ON UPPER(NUME)+STR(MBAC,5) TO INM.IDX UNIQUE
BROWSE
CLOSE DATABASES
```

Exemplul 5. Să se indexeze tabela ESTUD1 după câmpul numeric MATR, într-un fişier de index IMATR.IDX, numai pentru înregistrările care au câmpul MBAC (medie bacalaureat) >8.00:

```
USE ESTUD1
INDEX ON MATR TO IMATR.IDX FOR MBAC>8.0
BROWSE
CLOSE DATABASES
```

8.3. Accesul la date

În funcție de tipul accesului la date, comenzile sunt:

- pentru *acces secvențial* : GO, SKIP, LOCATE, CONTINUE.
- pentru *acces direct*: SEEK, caz în care tabela trebuie indexată.

Comanda GO

GO [RECORD] <*nr_înreg*> [IN <*nr_zonă_de_lucru*> | IN <*alias_tabelă*>]

sau

GO TOP | BOTTOM [IN <*nr_zonă_de_lucru*> | IN <*alias_tabelă*>]

Comanda are ca efect mutarea pointer-ului de pe înregistrarea curentă pe înregistrarea *nr_înreg*.

- IN *nr_zonă_de_lucru* – specifică zona de lucru în care este mutat pointer-ul.
- IN *alias_tabelă* – specifică alias-ul tabelei în care este mutat pointer-ul.
- TOP – poziționează pointer-ul pe prima înregistrare din fişier.
- BOTTOM – poziționează pointer-ul pe ultima înregistrare.

Funcția RECNO()

Returnează numărul curent al înregistrării pe care este poziționat pointer-ul în fişierul curent sau specificat. Apelul funcției

RECNO(<*nr_zonă_de_lucru*> | <*alias_tabelă*>)

unde: *nr_zonă_de_lucru* – reprezintă zona de lucru în care este activă tabela. Dacă s-a lansat o comandă SEEK pentru a căuta o înregistrare şi aceasta nu a fost găsită, se poate utiliza comanda GO RECNO(0) pentru a returna numărul înregistrării cu cea mai apropiată valoare căutată. Funcția RECNO(0) va returna valoarea 0 dacă nu s-a găsit nici o valoare apropiată.

Funcția RECCOUNT()

Returnează numărul de înregistrări din tabela curentă sau specificată. Apelul funcției

RECCOUNT(<nr_zonă_de_lucru> | <alias_tabelă>)

Funcția EOF()

Este funcția logică care returnează valoarea adevărat (**.T.**), dacă pointer-ul de fişier este poziționat pe ultima înregistrare, în fişierul curent. În caz contrar, valoarea returnată este fals (**.F.**). Apelul funcției

EOF ([<*nr_zonă_de_lucru*> | <*alias_tabelă*>])

Funcția BOF()

Este o funcția logică şi returnează valoarea adevărat (**.T.**), dacă pointer-ul de fişier este poziționat pe prima înregistrare, în fişierul curent. În caz contrar, valoarea returnată este fals (**.F.**). Apelul funcției

BOF ([<*nr_zonă_de_lucru*> | <*alias_tabelă*>])

Comanda SKIP

SKIP [<*nr_înreg*>] [IN <*nr_zonă_de_lucru*> | IN <*alias_tabelă*>]

unde: *nr_înreg* – specifică numărul de înregistrări peste care se mută pointer-ul în fişier. Dacă nu se specifică, pointer-ul se va muta la următoarea înregistrare. Dacă valoarea lui *nr_înreg* este pozitivă, pointer-ul se mută către sfârşitul fişierului. Dacă valoarea este negativă, pointer-ul se mută către începutul fişierului.

Comanda SET FILTER TO

Precizează condiția pe care trebuie să o îndeplinească înregistrările din tabela curentă pentru a putea fi accesate.

SET FILTER TO [<*expr_L*>]

unde: *expr_L* – condiția de accesare a înregistrărilor. Numai înregistrările care satisfac expresia logică *expr_L* sunt disponibile şi toate comenzile care vor accesa tabela, vor respecta condiția declarată în comanda SET FILTER TO. Dacă este omisă expresia logică, toate înregistrările sunt accesibile.

Comanda LOCATE

Realizează căutarea secvențială într-o tabelă, a primei înregistrări care îndeplineşte condițiile specificate în expresia logică de căutare, *expr_L* .

LOCATE FOR <*expr_L*_1> [*Scope*] [WHILE <*expr_L*_2>]

unde:

- FOR *expr_L* _1 este condiția pentru care are loc căutarea secvențială.
- *Scope* – specifică domeniul înregistrărilor în care are loc căutarea. Numai înregistrările din interiorul domeniului vor fi evaluate. Clauza poate lua valorile **ALL**, **NEXT** *n*, **RECORD** *n*, **REST**. Clauza implicită este **ALL** (toate înregistrările).
- WHILE *expr_L*_2 – specifică condiția pentru căutarea înregistrărilor, atâta vreme cât *expr_L*_2 este adevărată.

Comanda CONTINUE

Se utilizează după o comandă LOCATE, pentru a continua căutarea, după găsirea primei înregistrări care îndeplineşte condiţiile specificate. Comanda CONTINUE mută pointer-ul de fişier la următoarea înregistrare pentru care expresia logică din comanda LOCATE este adevărată.

Exemplu. Presupunem că există o tabelă CLIENTI şi trebuie să se afle numărul clienţilor care sunt din Franţa.

```
USE CLIENTI
STORE 0 TO TOTAL
LOCATE FOR ALLTRIM(UPPER(TARA))='FRANTA'
DO WHILE FOUND()
    TOTAL=TOTAL+1
    CONTINUE
ENDDO
?'TOTAL CLIENTI DIN FRANTA:  "+LTRIM(STR(TOTAL))
CLOSE DATABASES
```

Comanda SEEK

Caută într-o tabelă o înregistrare la prima apariţie, a cărei cheie de index se potriveşte cu expresia generală şi mută pointer-ul pe înregistrarea găsită.

SEEK <*expr_gen*> [ORDER *nr_index* | *nume_fişier*_IDX | [TAG] *nume_etichetă_index* [OF *nume_fişier*_CDX] [ASCENDING | DESCENDING]] [IN <*nr_zonă_de_lucru*> | IN <*alias_tabelă*>]

unde:

- *expr_gen* – specifică cheia de index pentru care se caută cu comanda SEEK;
- ORDER *nr_index* – specifică numărul fişierului index care conţine cheia de indexare. Dacă s-au deschis odată cu tabela şi fişierele de index, este numărul care indică poziţia din lista de fişiere de index;
- ORDER *nume_fişier*_IDX - specifică numele fişierului de index;
- ORDER [TAG] *nume_etichetă_index* [OF *nume_fişier*_CDX] – specifică o etichetă de index dintr-un fişier .CDX care conţine cheia de indexare. Numele etichetei de index (TAG) poate fi dintr-un fişier .CDX structural sau dintr-un fişier .CDX independent (nestructural);

- [ASCENDING | DESCENDING] – specifică dacă înregistrările sunt căutate în ordine ascendentă sau descendentă.

Exemplu. Să se indexeze fişierul ESTUD2 după câmpul NUME şi să se caute înregistrările care încep cu literele BAR. Structura tabelei ESTUD2 este: MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), DATAN D, MBAC N(5,2)

```
USE ESTUD2
INDEX ON UPPER(NUME) TAG NUME
STORE 'BAR' TO CAUTL
SEEK CAUTL
IF FOUND( )
    BROWSE FIELDS NUME, ADRESA FOR UPPER(NUME)=CAUTL
ELSE
WAIT WINDOW 'NU EXISTA '
ENDIF
CLOSE DATABASES
```

Funcția FOUND()

Returnează valoarea logică adevărat (**.T.**), dacă comenzile CONTINUE, LOCATE sau SEEK s-au executat cu succes.

FOUND(<*zonă_de_lucru*> | <*alias_tabelă*>)

Dacă argumentele *zonă_de_lucru*, *alias_tabelă* sunt omise, funcția va returna rezultatul pentru fişierul deschis în zona de lucru curentă.

Vizualizarea conținutului unei tabele

Comanda afişează informații (nume câmpuri, date) despre tabela activă în ferestra principală VFP sau într-o fereastră definită de utilizator. Sintaxa este

DISPLAY[[FIELDS] <*listă_câmpuri*>] [*Scope*] [FOR <*expr_L*_1>] [WHILE <*expr_L_2*>]
[NOCONSOLE] [TO PRINTER [PROMPT] | TO FILE <*nume_fişier*>]

unde:

- [FIELDS] <*listă_câmpuri*>] – specifică câmpurile care vor fi afişate. Dacă se omite clauza se vor afişa toate câmpurile.
- *Scope* – domeniul de adresare a înregistrărilor (**ALL**, **NEXT** *n*, **RECORD** *n*, **REST**);
- FOR <*expr_L*_1> - vor fi afişate doar înregistrările care satisfac condiția logică;
- WHILE <*expr_L*_2> - atâta vreme cât condiția este îndeplinită, înregistrările vor fi afişate.

Se mai poate utiliza şi comanda LIST care are aceleaşi clauze.

8.4. Actualizarea datelor

Operația de actualizare constă în adăugarea de noi înregistrări, modificarea înregistrărilor existente, ştergerea logică şi/sau fizică a înregistrărilor. Operațiile fac parte din limbajul de manipulare a datelor (LMD).

8.4.1. Adăgarea de noi înregistrări

Adăugarea se poate face utilizând comenzile de tip APPEND, prin preluarea datelor din alte tabele, masive de date şi variabile de memorie.

APPEND [BLANK] [IN *nr_zonă_de_lucru* | *alias_tabelă*]

Comanda are ca efect adăugarea unei înregistrări vide la sfârşitul unei tabele.

APPEND FROM <*nume_fişier*> [FIELDS <*listă_câmpuri*>] [FOR <*expr_L*>] [TYPE]
[DELIMITED [WITH <*delimitator*> | WITH BLANK | WITH TAB]]

Comanda are ca efect adăugarea unei înregistrări preluată din altă tabelă, sau dintr-un fişier de tip text, în tabela curentă, unde:

- *nume_fişier* – specifică numele fişierului din care se face adăugarea;
- FIELDS <*listă_câmpuri*> - specifică lista câmpurilor în care vor fi adăugate datele;
- FOR <*expr_L*> - se va adăuga câte o nouă înregistrare, din înregistrările tabelei selectate pentru adăugare, cele care îndeplinesc condiția dată de expresia logică *expr_L;*
- TYPE – specifică tipul fişierului sursă din care se adaugă. Se foloseşte în cazul în care nu este o tabelă VFP;

- DELIMITED – fişierul sursă din care se adaugă datele în tabela VFP este un fişier de tip ASCII în care fiecare înregistrare se termină cu <CR> şi <LF> (**Enter** şi **Line Feed** - salt la linie nouă). Câmpurile conţinute sunt implicit separate prin virgulă, iar cele de tip caracter sunt declarate între apostrof;
- DELIMITED WITH <*delimitator*> - câmpurile de tip carcater, sunt separate cu un caracter diferit de apostrof;
- DELIMITED WITH BLANK - câmpurile sunt separate cu spaţii în loc de virgulă;
- DELIMITED WITH TAB - câmpurile sunt separate cu cu TAB-uri în loc de virgulă.

APPEND FROM ARRAY <*nume_masiv*> [FOR <*expr_logică*>] [FIELDS <*listă_câmpuri*> | FIELDS LIKE <*şablon*> | FIELDS EXCEPT <*şablon*>]

Comanda are ca efect adăugarea unei înregistrări în tabela curentă, pentru fiecare linie de masiv, datele scrise în fiecare câmp corespund coloanelor din masiv. Semnificaţia parametrilor şi a clauzelor este următoarea:

- *nume_masiv* – specifică numele masivului care conţine datele ce vor fi copiate ca înregistrări noi;
- FOR <*expr_logică*> - specifică condiţia pentru adăugarea înregistrărilor în tabelă. Expresia logică trebuie să conţină şi o condiţie pentru un câmp al înregistrării. Înainte de a fi adăugată o linie din masiv în tabelă sub formă de înregistrare, în expresia logică se verifică dacă elementele liniei de masiv respectă condiţia;
- FIELDS <*listă_câmpuri*>- specifică faptul că numai anumite câmpuri (cele din listă) din noua înregistrare vor fi actualizate cu date din linia masivului;
- FIELDS LIKE <*şablon*> - specifică faptul că doar câmpurile care se potrivesc şablonului vor fi actualizate.
- FIELDS EXCEPT <şablon>- specifică faptul că toate câmpurile vor fi actualizate, cu excepţia celor care se încadrează în şablon.

Dacă masivul este unidimensional, comanda APPEND FROM ARRAY adaugă o singură înregistrare în tabelă, conţinutul primului element trece în primul câmp al înregistrării, conţinutul celui de al doilea element trece în cel de al doilea câmp etc., câmpurile MEMO şi GENERAL sunt ignorate.

Exemplu comanda:

APPEND FROM ARRAY NUME FIELDS LIKE M*, A* EXCEPT NUM*

are ca efect adăugarea din masivul NUME, a câmpurilor care încep cu litera M şi A mai puţin cele care au primele trei litere NUM. Caracterul asterisc (**Wildcard**) are semnificaţia de orice combinaţie de caractere.

Notă Utilizarea clauzelor FIELDS LIKE şi EXCEPT trebuie să se facă într-o tabelă ale cărui înregistrări au câmpuri care acceptă valori de tip **NULL** (pentru a putea fi exceptate de la scriere).

Există şi comanda inversă, cu care putem adăuga datele din înregistrarea curentă a unei tabele într-un masiv:

SCATTER [FIELDS <*listă_câmpuri*>] | [FIELDS LIKE <*şablon*>] | [FIELDS EXCEPT <*şablon*>] [MEMO] TO ARRAY <*nume_masiv*> [BLANK]

Semnificaţia clauzelor este aceaşi ca şi la comanda APPEND FROM ARRAY ..., clauza BLANK determină crearea masivului cu elemente vide, care sunt de aceaşi mărime şi tip cu câmpurile din tabelă. Clauza MEMO specifică existenţa unui câmp de tip MEMO în listă; implicit câmpurile MEMO sunt ignorate.

Exemplu. Să se creeze un masiv cu numele MASIV1, să se încarce cu date vide, de acelaşi tip şi mărime cu cele ale câmpurilor tabelei ESTUD1 (MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), DATAN D, MBAC N(5,2)). Să se introducă datele câmpurilor în masiv şi să se adauge sub formă de înregistrare nouă în tabela ESTUD1.

```
USE ESTUD1
* se deschide tabela ESTUD1
SCATTER TO MASIV1 BLANK
* se crează vectorul MASIV1, cu elemente vide şi de acelaşi tip şi mărime
* cu câmpurile din tabela ESTUD1
FOR I=1 TO ALEN(MASIV1)
@I,1 SAY FIELD(I) GET MASIV1(I)
```

```
* se afişează numele câmpului  şi se editează elementul I din masiv
ENDFOR
READ
* se citesc valorile introduse cu GET
IF READKEY()!=12
* dacă nu s-a apăsat tasta <ESC>
 APPEND FROM ARRAY MASIV1
* se adaugă din elementele masivului o nouă înregistrare în tabela
ENDIF
BROWSE
* vizualizare tabela ESTUD1
CLOSE TABLE
* închidere tabela ESTUD1
CLEAR
```

APPEND GENERAL <*nume_câmp_general*> FROM <*nume_fişier*> DATA <*expr_c*> [LINK] [CLASS <*nume_clasă*>]

Comanda realizează importul unui obiect OLE dintr-un fişier şi îl plasează într-un câmp de tip general. Semnificaţia parametrilor şi a clauzelor este următoarea:

- *nume_câmp_general* – specifică numele câmpului general în care obiectul OLE va fi plasat.
- FROM <*nume_fişier*> - specifică numele fişierului care conţine obiectul OLE;
- DATA <*expr_c*> - expresie de tip caracter care este evaluată şi trecută sub formă de şir de caractere către obiectul OLE din câmpul general. Obiectul OLE trebuie să fie capabil să primescă şi să proceseze şirul de caractere. De exemplu, nu poate fi trimis un şir de caractere către un obiect grafic creat cu Paint Brush;
- LINK - crează o legătură între obiectul OLE şi fişierul care conţine obiectul. Obiectul OLE apare în câmpul general, dar definirea obiectului rămâne în fişierul care-l conţine. Dacă se omite clauza, obiectul este înglobat în câmpul general;
- CLASS <*nume_clasă*> - specifică o clasă pentru un obiect de tip OLE, alta decât clasa implicită.

8.4.2. Modificarea înregistrărilor

Comanda CHANGE afişează câmpurile unei tabele pentru editare (modificare).

CHANGE [FIELDS <*listă_câmpuri*>][*Scope*] [FOR <*expresie_L*_1>] [WHILE <*expresie_L*_2>] [FONT <*nume_literă*> [,<*mărime_literă*>]] [FREEZE <*nume_câmp*>] [NOAPPEND] [NODELETE] [NOEDIT]

Semnificaţia clauzelor este următoarea*:*

- FIELDS <*listă_câmpuri*> - specifică câmpurile care vor apare în fereastra de editare;
- *Scope* – specifică domeniul de afişare a înregistrărilor (**ALL**, **NEXT** *n*, **RECORD** *n*, **REST**);
- FOR <*expresie_L*_1> - specifică faptul că doar înregistrările care satisfac condiţia logică dată de *expresie_L*_1, vor fi afişate în fereastra de editare;
- WHILE <*expresie_L*_2> - atâta vreme cât *expresie_L*_2 este adevărată vor fi afişate înregistrările în fereastra de editare pentru modificare;
- FONT <*nume_literă*> [,<*mărime_literă*>] – specifică pentru fereastra de editare, tipul de literă şi mărimea, cu care vor fi afişate datele;
- FREEZE <*nume_câmp*> - permite ca modificările să fie făcute doar în câmpul specificat cu *nume_câmp*. Celelalte câmpuri vor fi afişate dar nu pot fi editate (modificate);
- NOAPPEND – împiedică utilizatorul de a adăuga noi înregistrări (se blochează **Append Mode** din meniul sistem **View** sau combinaţia de taste **[Ctrl]** + **[Y]**);
- NODELETE – împiedică marcarea înregistrărilor pentru ştergere din interiorul ferestrei de editare. Includerea clauzei nu inhibă comanda de marcare pentru ştergere din interiorul unei proceduri;
- NOEDIT – împiedică un utilizator să modifice o tabelă. Includerea clauzei permite căutarea sau răsfoirea tabelei dar fără a-l putea modifica (edita).

Se poate folosi şi comanda EDIT cu aceiaşi parametri ca şi CHANGE.

Comanda BROWSE este una dintre cele mai utilizate comenzi pentru afişarea şi editarea înregistrărilor dintr-o tabelă. Cu ajutorul tastelor **[TAB]** sau **[SHIFT]** + **[TAB]** se poate face deplasarea în câmpul următor sau anterior. Cu tastele **[Page Up]**, **[Page Down]** se pot face deplasări pe verticală în tabelă.

Apăsarea simultană a tastelor: **[Ctrl]** + **[End]** sau **[Ctrl]** + **[W]** - ieşire cu salvare din fereastra de BROWSE; **[Ctrl]** + **[Q]** sau **[Esc]** ieşire fără salvarea modificărilor.

BROWSE [FIELDS <*listă_câmpuri*>] [FONT <*nume_font*>[,<*mărime_font*>]] [STYLE <*stil_font*>] [FOR <*expr_L*_1>] [FREEZE <*nume_câmp*>] [LOCK <*nr_câmpuri*>] [NOAPPEND] [NODELETE] [NOEDIT | NOMODIFY] [VALID <*expr_L*_2>] [ERROR <*mesaj*>] [WHEN <*expr_L*_3>] [WIDTH <*lățime_afişare_câmp*>] [IN [WINDOW] <*nume_fer*>]

unde:

- FIELDS <*listă_câmpuri*> - specifică care câmpuri vor aparea în fereastra BROWSE. Câmpurile vor fi afişate în ordinea specificată în listă;
- FONT <*nume_font*>[,<*mărime_font*>] – specifică tipul literei şi mărimea utilizate în fereastra **Browse** (de exemplu **Font** 'Courier', 16);
- STYLE <*stil_font*> - specifică stilul literelor folosite în fereastra **Browse** (**B**- îngroşat, **I**- italic, **U**- subliniat);
- FOR <*expr_L*_1> - vor fi afişate numai acele înregistrări care îndeplinesc condiția dată de expresia logică *expr_L*_1;
- FREEZE <*nume_câmp*> - permite modificări doar într-un singur câmp, specificat prin *nume_câmp*;
- LOCK <*nr_câmpuri*> - specifică numărul câmpurilor care pot fi afişate în partea stângă, în fereastra **Browse**, fără a fi necesară deplasarea cu **[TAB]**-uri sau cursorul ferestrei (**Scroll Bar**);
- NOAPPEND – împiedică utilizatorul să adauge noi înregistrări (se inhibă acțiunea tastelor **[Ctrl]** + **[Y]** şi opțiunea **Append** new record din meniul **Table**);
- NODELETE – împiedică marcarea înregistrărilor pentru ştergere în interiorul ferestrei BROWSE;
- NOEDIT | NOMODIFY – împiedică utilizatorul să facă modificări în fereastra **Browse**.
- VALID <**expr_L**_2> [ERROR <*mesaj*>] – realizează o validare la nivel de înregistrare. Clauza se execută doar dacă se fac modificări şi se încearcă trecerea la o altă înregistrare. Dacă *expr_L*_2 returnează valoarea **.T.** (adevărat) utilizatorul se poate deplasa la următoarea înregistrare. În caz contrar, se generează un mesaj de eroare;
- WHEN <*expr_L*_3> - se evaluaeză condiția dată de expresia logică *expr_L*_3 atunci când utilizatorul mută cursorul la o altă înregistrare. Dacă expresia returnează valoarea **.F.** (fals) înregistrarea nu mai poate fi modificată (devine **read-only**);
- WIDTH <*lățime_afişare_câmp*> - limitează numărul de caractere afişate pentru toate câmpurile din fereastră;

- IN [WINDOW] <*nume_fer*> - specifică fereastra părinte în interiorul căreia va fi deschisă fereastra **Browse**. Dacă fereastra părinte este mutată, se va muta şi fereastra **Browse**.

Comanda REPLACE realizează actualizarea câmpurilor dintr-o înregistrare.

REPLACE <*nume_câmp*_1> WITH <*expr*_1> [ADDITIVE] [,<*nume_câmp*_2> WITH <*expr*_2> [ADDITIVE]] ... [*Scope*] [FOR <*expr_L*_1>] [WHILE <*expr_L*_2>] [IN *nr_zonă_de_lucru* | *alias_tabelă*]

unde:

- <*nume_câmp*_1> WITH <*expr*_1> [,<*nume_câmp*_2> WITH <*expr*_2>] precizează că datele din *nume_câmp*_1 vor fi înlocuite cu valorile expresiei *expr*_1, datele din *nume_câmp*_2 cu valorile *expresiei expr*_2 etc.
- [ADDITIVE] – pentru câmpurile de tip memo, adaugă valoarea expresiei la sfârşitul câmpului memo (în continuare). Dacă clauza lipseşte, atunci câmpul memo va fi rescris cu valoarea dată de expresie;
- *Scope* – specifică domeniul înregistrărilor care vor fi înlocuite (**NEXT** *n*, **ALL**, **REST**, **RECORD** *n*);
- FOR <*expr_L*_1> - specifică faptul că vor fi înlocuite câmpurile desemnate, doar din înregistrările pentru care evaluarea expresiei logice *expr_L*_1 are valoarea **.T.** (adevărat);
- WHILE <*expr_L*_2> - precizează condiția pentru care atâta vreme cât expresia logică are valoarea **.T.**, câmpurile desemnate vor fi modificate;
- IN *nr_zonă_de_lucru* | *alias_tabelă* – specifică zona de lucru sau alias-ul tabelei, în care înregistrările vor fi actualizate.

Exemplu. În fişierul ESTUD2, să se modifice a treia înregistrare, câmpul ADRESA, cu o nouă valoare “Cluj”.

```
USE ESTUD2
REPLACE ADRESA WITH "Cluj" FOR RECNO()=3
BROWSE
CLOSE DATABASE
```

Comanda SET SAFETY ON | OFF determină afişarea de către sistemul VFP a unei casete de dialog înainte de a suprascrie un fişier.

SET SAFETY ON | OFF

- Clauza ON este implicită şi determină afişarea unei casete de dialog, pentru confirmarea de către utilizator a suprascrierii fişierului.
- Clauza OFF împiedică afişarea casetei de confirmare a suprascrierii.

8.4.3. Ştergerea înregistrărilor

Ştergerea înregistrărilor se poate face la nivel logic sau fizic.

Comanda DELETE realizează ştergerea la nivel logic. Înregistrările continuă să existe în fişier dar sunt marcate ca fiind şterse.

DELETE [*Scope*] [FOR <*expr_L*_1>] [WHILE <*expr_L*_2>] [IN <*nr_zonă_de_lucru*> | <*alias_tabelă*>]

unde:

- *Scope* – specifică domeniul în care se face marcarea pentru ştergere a înregistrărilor;
- FOR <*expr_L*_1> - specifică condiția pentru care vor fi marcate pentru ştergere acele înregistrări pentru care *expr_L*_1 este adevărată;
- WHILE <*expr_L*_2> - vor fi marcate pentru ştergere înregistrări, atâta vreme cât condiția *expr_L*_2 este adevărată.

Funcția DELETED() returnează o valoare logică care indică dacă înregistrarea curentă a fost marcată pentru ştergere.

DELETED ([*nr_zonă_de_lucru* | *alias_tabelă*])

Comanda RECALL deselectează înregistrările marcate pentru ştergere.

RECALL [*Scope*] [FOR <*expr_L*_1>] [WHILE <*expr_L*_2>]

unde:

- *Scope* – precizează domeniul în care se aplică deselectarea;

- FOR <*expr_L*_1> - doar înregistrările pentru care *expr_L*_1 este adevărată (**.T.**) vor fi deselectate;
- WHILE <*expr_L*_2> - atâta vreme cât *expr_L*_2 este adevărată, înregistrările vor fi deselectate.

Comanda PACK face parte din comenzile de ştergere la nivel fizic; va şterge toate înregistrările marcate pentru ştergere din tabela curentă.

PACK [MEMO] [DBF]

unde:

- MEMO – permite ştergerea fişierului MEMO ataşat tabelei curente, dar nu şi înregistrările marcate pentru ştergere (fişierul MEMO are acelaşi nume cu tabela, dar extensia este .FPT);
- DBF – se vor şterge înregistrările marcate pentru ştergere din tabela curentă, dar nu afectează fişierul MEMO.

Comanda ZAP şterge fizic toate înregistrările din tabelă, rămânând doar structura tabelei. Este echivalentă cu comanda DELETE ALL, urmată de comanda PACK, dar este mult mai rapidă.

ZAP [*nr_zonă_de_lucru* | *alias_tabelă*]

Comanda DELETE FILE şterge un fişier de pe disc.

DELETE FILE [<*nume_fişier*>] [RECYCLE]

unde:

- *nume_fişier* – specifică numele fişierului şi extensia. Numele fişierului poate conține şi caracterul asterisc: DELETE FILE *.BAK va avea ca rezultat ştergerea tuturor fişierelor cu extensia .BAK de pe disc. Comanda se execută pentru fişierele care nu sunt deschise (active);
- RECYCLE – fişierul nu va fi şters imediat de pe disc ci mutat în directorul RECYCLE BIN din sistemul WINDOWS 9x. Se mai poate folosi şi comanda ERASE care are aceleaşi clauze.

Comanda SET DELETED specifică dacă sistemul VFP va procesa (şterge) înregistrările marcate pentru ştergere sau dacă acestea sunt disponibile pentru a fi utilizate în alte comenzi.

SET DELETED ON | OFF

Semnificaţia clauzelor:

- Clauza OFF este implicită. Înregistrările marcate pentru ştergere pot fi accesate de alte comenzi care operează cu înregistrări;
- Clauza ON are ca efect ignorarea înregistrărilor marcate pentru ştergere, de comenzile care operează cu înregistrări.

Comanda COPY TO crează un nou fişier cu acelaşi conţinut ca şi cel al tabelei curente.

COPY TO <*nume_fişier*> [FIELDS <*listă_câmpuri*> | FIELDS LIKE <*şablon*> | FIELDS EXCEPT <*şablon*>][*Scope*] [FOR <*exprL*_1>] [WHILE <*exprL*_2>] [[WITH] CDX] | [[WITH] PRODUCTION]

unde:

- *nume_fişier* – specifică numele noului fişier care se crează cu comanda COPY TO;
- FIELDS <*listă_câmpuri*> - specifică în listă câmpurile ce vor fi copiate în noul fişier. Dacă clauza se omite, atunci toate câmpurile din tabela activă vor fi copiate. Prin declararea clauzei FIELDS se realizează operaţia de proiecţie (se selectează numai anumite câmpuri - coloane - din tabelă);
- FIELDS LIKE <*şablon*> - specifică câmpurile din tabela sursă care se potrivesc unui şablon şi vor fi copiate în noul fişier. Se poate folosi şi caracterul asterisc (de exemplu comanda COPY TO TABELA1 FIELDS LIKE P*, A* va avea ca efect copierea în noul fişier a tuturor câmpurilor care încep cu litera P şi A);
- FIELDS EXCEPT <*şablon*> - vor fi copiate toate câmpurile, mai puţin cele care se încadrează în şablonul specificat;
- *Scope* – specifică domeniul din care vor fi selectate înregistrările care vor fi copiate.
- FOR <*exprL*_1> - vor fi copiate doar acele înregistrări pentru care evaluarea expresiei logice exprL_1 are valoarea adevărat (**.T.**)
- WHILE <*exprL*_2> - specifică condiţia pentru care atâta vreme cât expresia logică *exprL*_2 este adevărată, înregistrările vor fi copiate în noul fişier.

Notă Dacă se utilizează clauzele [*Scope*], FOR sau WHILE, fără clauza FIELDS, se va implementa operatorul relațional de selecție (se aleg doar o parte din înregistrări, cu toate câmpurile). Dacă se folosește clauza FIELDS în combinație cu clauzele [*Scope*], FOR sau WHILE, va rezulta o operație compusă: proiecție și selecție.

- [WITH] CDX] | [[WITH PRODUCTION] – la copiere se va crea și un fișier de index structural pentru noua tabelă, care este identic cu cel al tabelei sursă. Etichetele de index (TAG-urile) vor fi copiate în noul fișier de index structural. Cele două clauze sunt identice. Clauzele nu se folosesc dacă se crează un fișier care nu este tabelă VFP (extensia .DBF).

Comanda SET ALTERNATE direcționează la ecran sau la imprimantă, ieșirea rezultată din folosirea comenzilor DISPLAY sau LIST.

SET ALTERNATE ON | OFF

sau

SET ALTERNATE TO [<*nume_fișier*> [ADDITIVE]]

unde:

- ON – direcționează ieșirea către un fișier de tip text;
- OFF – clauza implicită, dezactivează ieșirea către un fișier de tip text;
- TO *nume_fișier* – crează un fișier de tip text, cu extensia implicită .TXT ;
- ADDITIVE – clauză prin care ieșirea este adăugată la sfârșitul fișierului specificat cu *nume_fișier*. Dacă se omite clauza, conținutul fișierului de ieșire este suprascris.

Comanda CLOSE ALTERNATE închide un fișier deschis cu comanda SET ALTERNATE. Sintaxa comenzii este

CLOSE ALTERNATE

8.4.5. Relații între tabele

Comanda SET RELATION TO stabilește o legătură între două tabele deschise.

SET RELATION TO [<*expresie*1> INTO <*nr_zonă_de_lucru*> | <*alias_tabelă*> [, <*expresie*2> INTO <*nr_zonă_de_lucru*> | <*alias_tabelă*>...] [IN <*nr_zonă_de_lucru*> | <*alias_tabelă*>] [ADDITIVE]]

unde:

- *expresie*1 – specifică expresia relațională care stabileşte legătura între tabela-părinte şi tabela-fiu. Expresia relațională este de obicei câmpul indexat din tabela-fiu. Indexul pentru tabela-fiu poate proveni dintr-un fişier simplu de index (.IDX) sau un fişier de index compus (.CDX);
- INTO *nr_zonă_de_lucru* | *alias_tabelă* – specifică numărul zonei de lucru sau alias-ul tabelei-fiu;
- IN *nr_zonă_de_lucru* | *alias_tabelă* – specifică zona de lucru sau alias-ul tabelei-părinte;
- ADDITIVE – clauza păstrează toate celelalte relații stabilite anterior în zona de lucru specificată.

Pentru a stabili o relație între două tabele trebuie îndeplinite următoarele condiții:

- cele două tabele între care se stabileşte relația tata-fiu trebuie să fie indexate după aceeaşi cheie (câmp), care face legătura între ele;
- cele două tabele trebuie deschise în zone de lucru diferite.

Exemplul 1.

Tabelele legate printr-o relație au în general un câmp comun. De exemplu presupunem că avem o tabelă STUDENT, care conține informații despre studenți şi are câmpuri pentru *nume*, *adresă*, şi un cod unic al studentului (*matr*). O a doua tabelă, NOTE, conține informații despre notele studentului (*matr*, *cod_disc*, *nota*), care va avea de asemenea un câmp pentru codul unic al studentului (*matr*). Se pot uni astfel informațiile despre studenți cu cele despre note. Comanda SET RELATION leagă cele două tabele prin câmpul comun *matr*. Pentru a stabili relația, tabela-fiu (NOTE) trebuie să fie indexat după câmpul comun. După stabilirea relației, de câte ori mutăm pointer-ul pe o înregistrare în fişierul STUDENT, pointer-ul din fişierul NOTE (fiu) se va muta pe înregistrarea care are acelaşi cod *matr* corespunzător în fişierul STUDENT.

Fişier **discipline.dbf**		Fişier **student.dbf**		Fişier **note.dbf**	
Cod_disc	N(3)	Matr	N(5)	Matr	N(5)
Denumire	C(60)	Nume	C(50)	Cod_disc	N(3)
Exam	L	Adresa	m	Nota	N(2)
Coloc	L	An_stud	C(2)		
Tip_curs	C(1)	Grupa	N(2)		
		Media	N(5,2)		

Integr L

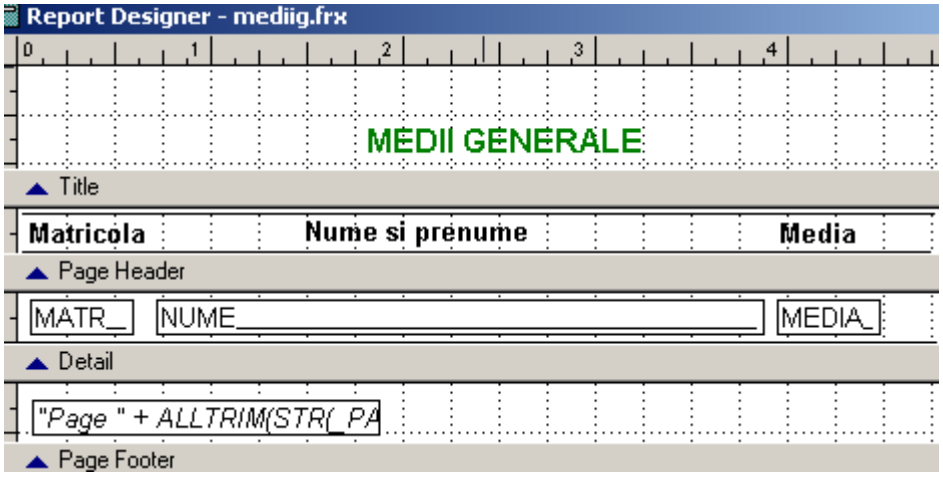
Programul pentru calculul mediilor generale

```
PUBLIC MEDG, INTEGRAL
SELECT 2
USE STUDENT
SELECT 3
USE NOTE
FOR I=1 TO RECCOUNT(2)
    GOTO I
    MATRICOLA=MATR
     INTEGRAL=.T.
    MEDG=0
     K=0
    SELECT 3
    LOCATE FOR MATR=MATRICOLA
    DO WHILE FOUND(3)
        K=K+1
         MEDG=MEDG+NOTA
        IF NOTA<5 THEN
            INTEGRAL=.F.
         ENDIF
        CONTINUE
    ENDDO
    MEDG=MEDG/K
    SELECT 2
    REPLACE MEDIA WITH MEDG, INTEGR WITH INTEGRAL
ENDFOR
SELECT 2
SORT TO TEMP1 ON NUME/A
CLOSE DATABASE
DELETE FILE 'STUDENT.DBF'
RENAME 'TEMP1.DBF' TO 'STUDENT.DBF'
SELECT 2
```

USE STUDENT

REPORT FORM MEDIIG PREVIEW

CLOSE DATABASES



Programul pentru stabilirea relațiilor

SELECT 2

USE STUDENT

SELECT 3

USE DISCIPLINE

SELECT 4

USE NOTE

SELECT 2

INDEX ON MATR TAG MATR ADDITIVE

&& se indexează după câmpul matr

SELECT 3

INDEX ON COD_DISC TAG COD_DISC

&& se indexează după câmpul cod_disc

SELECT 4

SET ORDER TO TAG MATR OF STUDENT.CDX IN STUDENT

&& se stabileşte indexul principal

SET RELATION TO MATR INTO STUDENT ADDITIVE

&& SE stabileşte relația între student şi note

SET ORDER TO TAG COD_DISC OF DISCIPLINE.CDX IN DISCIPLINE

&& se stabileşte indexul principal

SET RELATION TO COD_DISC INTO DISCIPLINE ADDITIVE
&& se stabileşte relaţia între discipline, note
REPORT FORM REPORT1 PREVIEW
SET RELATION OFF INTO STUDENT
SET RELATION OFF INTO DISCIPLINE
CLOSE DATABASE

Pentru crearea raportului REPORT1.FRX se deschid tabelele: STUDENT, NOTE, DISCIPLINE, se accesează meniul **File/New** opţiunea **Report.**

Pagina de raport este structurată în trei secţiuni: **Page Header** (antet), **Detail** (conţinut), **Page Footer** (subsol pagină). Datele pot fi grupate după una sau mai multe variabile, introducându-se în **GroupHeader**; **Detail** sau **Group Footer** (Figura 1).

In raport se includ următoarele câmpuri de date (Figura 1):

- *recno*() – numărul înregistrării din tabelă,
- *student.nume*,
- *student.media*,
- *note.nota*,
- *discipline.denumire.*

Raportul poate conţine şi elemente grafice: linii / chenare utilizate pentru construcţia de tabele sau imagini.

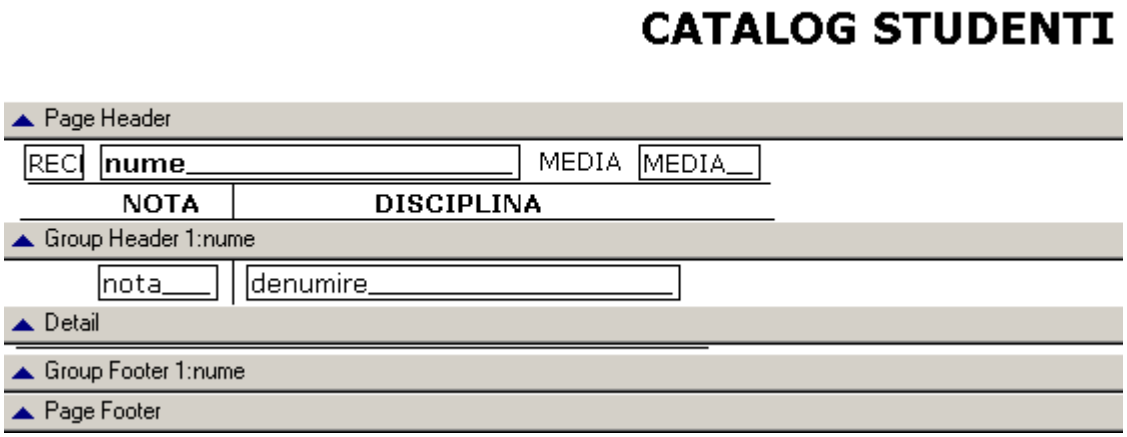


Fig. 1

Exemplul 2.

Presupunem că avem o tabelă CLIENŢI, care conţine informaţii despre clienţi, câmpuri pentru *nume*, *adresă*, şi un cod unic al clientului (*cod_client*). O a doua tabelă, FACTURI, conţine

informații despre facturi (*număr*, *cantitate*, *produse* etc.), care va avea de asemenea un câmp pentru codul clientului. Se pot uni astfel informațiile despre clienți cu cele despre facturi. Comanda SET RELATION leagă cele două tabele prin câmpul comun *cod_client*. Pentru a stabili relația, tabela-fiu (FACTURI) trebuie să fie indexat după câmpul comun. După stabilirea relației, de câte ori se mută pointer-ul în fişierul CLIENȚI pe o înregistrare, pointer-ul din fişierul FACTURI (fiu) se va muta pe înregistrarea care are acelaşi *cod_client* în fişierul CLIENȚI.

```
SELECT 2
USE CLIENTI
SELECT 3
USE FACTURI
INDEX ON COD_CLIENT TAG COD_CLIENT ADDITIVE
&& se indexează câmpul cod_client
SELECT 2
SET ORDER TO TAG COD_CLIENT OF FACTURI.CDX IN FACTURI
&& se stabileşte indexul principal
SET RELATION TO COD_CLIENT INTO FACTURI  ADDITIVE
&& se stabileste relatia intre clienți, facturi
```

Comanda RENAME redenumeşte un fişier.

RENAME <*nume_fişier_vechi*> TO <*nume_fişier_nou*>

unde:

- *nume_fişier_vechi* TO *nume_fişier_nou*2 – specifică numele fişierului sursă şi noul nume. Se va include extensia pentru fiecare fişier, dacă nu, implicit se va atribui extensia .DBF. Comanda RENAME nu se foloseşte pentru o tabelă care aparține unei baze de date. Pentru a schimba numele unei tabele dintr-o bază de date se foloseşte comanda RENAME TABLE. Numele fişierului sursă şi cel al fişierului nou creat poate cuprinde şi caracterul asterisc (RENAME *.PRG TO *.BAK).

Comanda DISPLAY FILES

Afişează informații despre un fişier.

DYSPLAY FILES [ON <*drive*>] [LIKE <*şablon*>] [TO PRINTER [PROMPT]] | TO FILE <*nume_fişier*>]

Unde:

- ON <*drive*> - specifică calea către fişiere;
- LIKE <*şablon*> - specifică condiția pentru care vor fi afişate informații, doar despre acele fişiere care se încadrează în şablon;
- TO PRINTER [PROMPT] | TO FILE <*nume_fişier*> - direcționează informațiile la imprimantă (cu afişarea unei casete de dialog, clauza PROMPT) sau într-un fişier.

Informații identice se pot obține şi cu comanda DIR care are aceleaşi clauze.

9. Programarea procedurală

Mediul VFP oferă pe lângă modul de lucru în fereastra de comandă (stil interpretor de comenzi) şi posibilitatea de a dezvolta programe sursă în limbajul propriu sistemului (stil compilator), prin comenzi şi funcţii care descriu datele şi pot efectua prelucrările necesare dezvoltării aplicaţiei.

Programul sursă foloseşte atât tehnica de programare procedurală (structurată, modulară) cât şi cea de programare pe obiecte. Într-un program sursă pot fi utilizate de asemenea şi comenzile din nucleul SQL, care permit o programare descriptivă şi de manipulare a datelor la nivel de tabelă.

9.1. Programarea structurată

Maniera de programare care foloseşte numai cele trei structuri de bază: structura liniară, structura alternativă şi structura repetitivă, este cunoscută sub numele de programare structurată. Se poate demonstra ca orice program poate fi realizat utilizând numai cele trei structuri de bază

Limbajul VFP are comenzi specifice pentru implementarea celor trei structuri de program fundamentale: *secvenţială, alternativă* (IF şi CASE) şi *repetitivă* (WHILE, FOR şi SCAN), dar nu are comenzi pentru salt necondiţionat.

Tehnica *programării modulare* se utilizează la rezolvarea unor probleme complexe şi implică descompunerea problemei în părţi componente (module), conform unei metode (top-down, bottom-up) şi analiza lor, care stabileşte:

- caracteristicile principale ale fiecărui modul;
- legăturile dintre module;
- alegerea unui modul principal (program, meniu sau videoformat);
- ierarhizarea modulelor faţă de ansamblu.

Aceste subprobleme sunt apoi programate utilizând tehnica programării structurate.

Limbajul VFP oferă facilitatea de creare a unui proiect (§12.2. Figura 1), în care aceste module pot fi declarate în una din categoriile implementate:

- **Data** – se pot declara baze de date (**Databases**), tabele libere (**Free Tables**), interogări (**Query**);
- **Class Libraries** – librării de clase;
- **Documents** - se pot declara videoformate (**Forms**), rapoarte (**Reports**), etichete (**Labels**);
- **Code** – programe (**Programs**), aplicaţii (**Applications**);
- **Other** – alte componente: meniuri (**Menus**), fişiere de tip text (**Text Files**);
- **Other Files** – alte fişiere utilizate: icon-uri, fişiere .BMP etc.

Toate aceste componente sunt înglobate în proiect, care constituie baza pentru crearea aplicației în format executabil.

9.1.1. Structura liniară

O structură liniară este constituită dintr-o succesiune de comenzi, dintre cele admise şi în modul de lucru interpretor (în fereastra de comenzi). O astfel de structură este o secvență de comenzi care nu conține structuri alternative sau repetitive, ci doar comenzi, funcții şi atribuiri de valori unor variabile.

Comenzi de afişare/citire la monitor

Comanda utilizată pentru afişarea la monitor, a unei variabile/constante de tip caracter sau numeric este:

@<*linie*>,<*coloana*> SAY <*expr*> [PICTURE <*exp_c*1>] | [FUNCTION <*exp_c*2>] [FONT <*exp_c*3>[,<*exp_n*>]] [STYLE <*exp_c*4>]

unde:

- @<*linie*>,<*coloana*> - caracterul @ fixează coordonatele de afişare, date sub formă de linie şi coloană.
- SAY <*expr*> - comanda de afişare pe ecran, la coordonatele fixate, a unei expresii sau a unei funcții.
- PICTURE <*expr_c*1> - formatul de afişare (se utilizează codurile din tabelă indicându-se formatul între apostrof: '9999.99', pentru date de tip numeric).
- FUNCTION<*exp_c*2> - oferă o alternativă la stabilirea formatului.
-

exp_c2	**Semnificație**
A	Caractere alfabetice
L	Date de tip logic (**.T.**, **.F.**, **.Y.**, **.N.**)
N	Litere şi cifre
X	Orice caractere
9	Cu date de tip caracter permite numai numere. Cu date de tip numeric permite numere şi semne algebrice

#	Permite cifre, semne algebrice şi spații
$	Afişează semnul curent al monedei (definit cu SET CURRENCY)

- FONT <*exp_c*3>[,<*exp_n*>] – specifică numele tipului de literă (font) utilizat şi mărimea acesteia.
- [STYLE <*exp_c*4> - definirea stilului de afişare (B – bold, I – italic, N – normal, U - subliniat).

Exemplu. Următoarea secvență de comenzi:

```
@1,1 SAY 'TEST' FUNCTION 'A' FONT 'ARIAL',14 STYLE 'BIU'
S=10
@3,1 SAY S PICTURE '99.99'
WAIT WINDOW  && afişarea unei ferestre de continuare
CLEAR
```

are ca efect afişarea în spațiul de lucru, în linia 1, coloana 1, a cuvântului TEST (îngroşat, înclinat, subliniat), iar în linia 3, coloana 1, valoarea variabilei *S* cu formatul ales.

Se pot afişa în spațiul de lucru sau tipări obiecte de tip BMP (imagini) cu comanda

@<*linie*>,<*coloana*> SAY <*fişier_imag*>BITMAP|<*câmp_GEN*> [STYLE <*exprC*>] [CENTER] [ISOMETRIC] | [STRETCH] [SIZE <*expN*1>,<*expN*2>]

unde:

- <*fişier_imag*>BITMAP|<*câmp_GEN*> - clauza care face referire la fişierul de tip BMP (Bitmap Picture) sau imaginea conținută într-un câmp general dintr-un fişier. <*fişier_imag*> este o expresie de tip caracter, delimitată de apostrof sau ghilimele;
- STYLE <*exprC*> – asignează atributul de opac – **Q** sau transparent – **T**, pentru obiectele inserate;
- CENTER – clauza pentru plasarea obiectului BMP în centrul unei arii determinate prin clauza SIZE sau delimitate de coordonatele linie, coloană;
- ISOMETRIC | STRETCH – clauze pentru cazul în care zona de afişare a obiectului BMP este mai mică decât dimensiunea reală a acestuia. ISOMETRIC redimensionează obiectul la dimensiunea ariei, cu păstrarea proporțiilor. STRECH crează un raport separat pentru fiecare coordonată *Ox*, *Oy,* cu care obiectul se poate distorsiona pe verticală sau orizontală.

- SIZE <*expN*1>,<*expN*2>] – stabileşte coordonatele colțului din dreapta jos al zonei de afişare.

Exemplu.

```
@1,1 SAY "1.JPG" BITMAP CENTER ISOMETRIC
WAIT WINDOW
CLEAR
```

Comanda

@<*linie*>,<*coloana*> GET <*var*> [PICTURE <*expr_c*>]

permite editarea variabilei de memorie *var* (pentru a introduce o nouă valoare), unde:

- PICTURE <*expr_c*> - reprezintă formatul de editare.

Se poate combina operația de afişare cu cea de editare a unei variabile, caz în care vom utiliza comanda:

@<*linie*>,<*coloana*> SAY <*mesaj*> GET <*var*>

Întotdeauna o comandă (sau mai multe comenzi) @...GET va fi însoțită de o comandă READ. Practic se poate construi o machetă cu câmpuri de afişare @…SAY şi câmpuri de citire @…GET, cursorul se deplasează de la o linie la alta apasând tasta **[ENTER]** sau **[TAB]** şi înapoi cu **[SHIFT]** + **[TAB]** (permite reactualizarea unor câmpuri înainte de citirea efectivă). La apăsarea tastei **[ENTER]** la ultima linie GET se va executa comanda READ şi se va citi întreg ecranul.

Comanda READ realizează citirea tuturor câmpurilor editate cu comanda GET.

READ [CYCLE] [MODAL] [VALID <*expr_L*>|<*expr_N*>] [WHEN <*expr_L*1>]

unde:

- CYCLE – clauză care împiedică încheierea citirii variabilelor editate, atunci când cursorul depăşeşte primul sau ultimul obiect creat cu GET. Dacă utilizatorul se plasează pe ultimul obiect GET şi apasă **[ENTER]** sau **[TAB]**, cursorul se va deplasa pe primul obiect GET. Pentru a încheia o comandă READ ciclic, se apasă tasta **[Esc]** sau tastele **[CTRL]** + **[W]** ;
- MODAL – este clauza care previne activarea ferestrelor care nu sunt implicate în execuția comenzii READ;

- VALID *expr_L* | *expr_N* – clauză care validează valorile introduse în câmpul GET, la terminarea comenzii READ. Dacă expresia logică returnează valoarea adevărat (**.T.**) comanda READ se încheie;
- WHEN *expr_L*1 – clauza condiționează execuția comenzii READ de evaluarea expresiei logice. Dacă expresia returnează valoarea fals comanda READ este ignorată.

9.1.2. Structura alternativă

Structura alternativă este implementată în cele două forme:

- cu două ramuri (comenzile IF, IIF);
- cu mai multe ramuri (comanda DO CASE).

Comanda IF…ENDIF execută condiționat un set de comenzi, în funcție de valoarea logică returnată de o expresie evaluată. Expresia evaluată nu furnizează decât două alternative (ramuri) corespunzătoare valorilor adevărat (THEN) sau fals (ELSE).

IF <*exprL*> [THEN] <*comenzi*> [ELSE <*comenzi*>] ENDIF

unde:

- *exprL* – specifică expresia logică ce este evaluată. Dacă cele două clauze THEN şi ELSE sunt prezente şi expresia este adevărată vor fi executate comenzile de pe ramura THEN, dacă expresia este falsă, vor fi executate comenzile de pe ramura ELSE;
- Dacă expresia logică este falsă şi ramura ELSE nu este inclusă, va fi executată prima comandă care urmează clauzei ENDIF;
- Clauza ENDIF trebuie inclusă la orice declarare a unui IF. O comandă IF poate include mai multe blocuri IF…ENDIF pe ramurile sale (imbricare), cu condiția ca închiderea lor (cu ENDIF) să se facă în aceaşi ordine ca şi deschiderea (nu se admite intersecția blocurilor).

Exemplu. În fişierul PRODUSE, să se caute un anumit produs; dacă este găsit să se afişeze câmpurile PRODUS, CANT, PU, dacă nu, să se afişeze un mesaj.

```
CLEAR
USE PRODUSE
GET EXPR 'INTRODUCETI CONDITIA DE LOCALIZARE' TO
GTEMP;
```

```
TYPE 'L' DEFAULT 'PRODUS = """
LOCATE FOR &gctemp
&& caută produsul cu comanda LOCATE

IF FOUND( )
&& dacă este găsit
DISPLAY FIELDS PRODUS,CANT,PU&& afişează câmpurile PRODUS, CANT, PU
ELSE
&& dacă nu
  ? 'CONDITIA ' + GTEMP + ' NU A FOST GASITA '
&& afişează mesajul
ENDIF
CLOSE TABLE
```

Funcția IIF returnează pe baza evaluării unei expresii logice, una din cele două valori ale unei expresii de tip numeric, caracter, monedă, dată calendaristică sau timp.

IIF(*expr_L*, *expr*_1, *expr*_2)

unde: *expr_L* – specifică expresia logică pe care funcția o evaluează.

- Dacă expresia logică este adevărată, va returnată valoarea expresiei *expr*_1.
- Dacă expresia logică este falsă va fi returnată valoarea dată de *expr*_2.

Această funcție este cunoscută şi sub numele de IF scurt (imediat). Este folosită mai ales la expresiile care condiționează afişarea conținutului unui câmp, în rapoarte sau etichete. Execuția sa este mai rapidă decât echivalentul său IF…ENDIF.

Comanda DO CASE…ENDCASE

Execută primul set de comenzi ale căror expresie condițională este adevărată. Comanda este echivalentul unor instrucțiuni IF în cascadă.

```
DO CASE
    CASE <expr_L_1>
        <comenzi1>
    CASE <expr_L_2>
        <comenzi2>
```

....

CASE *<expr_L_n>*

<comenzi_n>

[OTHERWISE

<comenzi>]

ENDCASE

Acțiunea comenzii:

- CASE *<expr_L_*1> *<comenzi*1>... - când prima expresie logică CASE întâlnită, are valoarea adevărat, blocul de comenzi va fi executat până la apariția unei noi comenzi CASE sau a clauzei ENDCASE. Execuția programului continuă cu prima instrucțiune executabilă care urmează lui ENDCASE;
- Dacă expresia din prima clauza CASE este falsă, blocul de comenzi aferent este ignorat şi se trece la evaluarea expresiei logice a următoarei clauze CASE;
- După găsirea primei expresii logice adevărată, orice altă clauză CASE pentru care expresia logică este adevărată, va fi ignorată;
- OTHERWISE *<comenzi>* - dacă toate expresiile logice CASE au returnat valoarea fals, clauza oferă o alternativă pentru introducerea unui bloc de comenzi.

Exemplu

```
@1,1 SAY "DATI N=" GET N
READ
DO CASE
    CASE N = 0
        ? 'ZERO'
    CASE N > 0
        ? 'POZITIV'
    OTHERWISE
        ? 'NEGATIV'
ENDCASE
WAIT WINDOW "Apasati Enter"
```

9.1.3. Structura repetitivă

Pentru descrierea structurilor repetitive limbajul pune la dispoziție următoarele tipuri de comenzi:

- ciclul de repetiție cu număr finit de paşi - comanda FOR...ENDFOR;
- ciclu cu număr nedefinit de paşi, condiționat anterior - comanda DO WHILE...ENDDO;
- o comandă de parcurgere a înregistrărilor unei tabele de la început la sfârşitul fişierului – comanda SCAN...ENDSCAN.

Comanda FOR...ENDFOR execută un bloc de comenzi de un număr de ori precizat.

FOR <*var*>=<*val_inițială*> TO <*val_finală*> [STEP <*val_increment*>] <*comenzi*> [EXIT] [LOOP] ENDFOR | NEXT

unde:

- *var* – specifică o variabilă sau un element de masiv, de tip întreg, care acționează ca un contor (variabilă de ciclare);
- *val_inițială*, *val_finală* – valoarea inițială, respectiv finală a contorului;
- STEP <*val_increment*> - domeniul dat de valoarea inițială / finală poate fi parcurs cu incrementul 1 (implicit) sau cu un pas (increment) precizat prin *val_increment*. Dacă *val_increment* este negativ contorul este decrementat;
- comenzi – specifică blocul de comenzi care vor fi executate în interiorul ciclului.
- EXIT – permite ieşirea forțată din ciclu (înainte de atingerea valorii finale a variabilei de ciclare). În program va fi executată prima comandă care urmează clauzei ENDFOR;
- LOOP – returnează controlul comenzii FOR, fără a se mai executa comenzile dintre LOOP şi ENDFOR. Contorul este incrementat sau decrementat ca şi cum s-ar ajunge la clauza ENDFOR.

Exemplu. Din fişierul CLIENTI, se vor afişa la monitor, din primele 10 înregistrări, din doi în doi, clienții.

```
USE CLIENTI
STORE 2 TO VI,K          && valoarea inițială şi pasul contorului
STORE 10 TO VF           && valoarea finală a contorului
FOR I=VI TO VF STEP K    && I  variabila contor
    GOTO I               && poziționare pe înregistrarea I
    DISPLAY FIRMA        && se afişează numele firmei de pe poziția I din fişierul CLIENȚI
```

```
ENDFOR                    && sfârşitul ciclului FOR
CLOSE TABLE
```

Comanda DO WHILE ... ENDDO execută un bloc de comenzi în interiorul unui ciclu condiționat anterior.

DO WHILE <*expr_L*> <*comenzi*> [LOOP] [EXIT] ENDDO

unde:

- *expr_L* – specifică o expresie logică a cărei valoare de adevăr determină dacă blocul de comenzi cuprins între DO WHILE şi ENDDO va fi executat. Atâta vreme cât condiția logică este adevărată, comenzile vor fi executate;
- LOOP – returnează controlul comenzii DO WHILE. Comenzile între LOOP şi ENDDO nu vor fi executate;
- EXIT – transferă controlul, din interiorul ciclului DO WHILE, către prima comandă care urmează clauzei ENDDO. Clauza este echivalentă cu o ieşire forțată din ciclu;
- Fiecare comandă DO WHILE trebuie să aibă un corespondent (să se închidă) ENDDO.

Exemplu. Din fişierul PRODUSE să se afişeze totalul produselor în stoc, care au prețul mai mare de 30$.

```
CLEAR
USE PRODUSE
STOC=0
DO WHILE !EOF()
    IF PU<30
        SKIP
    ENDIF
    STOC=STOC+CANT
    ? PRODUS,CANT
    SKIP
ENDDO
CLOSE TABLES
?STOC
```

Comanda SCAN...ENDSCAN realizează mutarea pointer-ului prin toate înregistrările din tabela curentă şi execută un bloc de comenzi pentru înregistrările care corespund condiţiilor specificate.

SCAN [*Scope*] [FOR <*expr_L_*1>] [WHILE <*expr_L_*2>] [*comenzi*] [LOOP] [EXIT] [ENDSCAN]

unde:

- *Scope* – specifică domeniul înregistrărilor care vor fi scanate (clauze: **ALL**, **NEXT** *n*, **RECORD** *n*, **REST**). Clauza implicită este **ALL**;
- FOR *expr_L_*1 – execută comenzile doar pentru înregistrările pentru care expresia logică *expr_L_*1 este adevărată. Includerea clauzei permite filtrarea înregistrărilor, eliminând pe cele pe care nu trebuie să fie scanate;
- WHILE *expr_L_*2 – specifică o condiţie pentru care blocul de comenzi va fi executat atâta vreme cât valoarea expresiei logice *expr_L_*2 este adevărată;
- LOOP – returnează controlul înapoi la comanda SCAN (se reia execuţia comenzii);
- EXIT – transferă programului controlul din interiorul buclei SCAN...ENDSCAN, la prima comandă care urmează clauzei ENDSCAN;
- ENDSCAN – indică sfârşitul procedurii de scanare.

Exemplu. Utilizând comanda SCAN, să se afişeze din fişierul CLIENŢI, numele firmei, oraşul şi persoana de contact, cu care o firmă are contracte în FRANŢA.

```
USE CLIENTI
SCAN FOR UPPER(TARA)='FRANTA'
    ? FIRMA,ORAS,CONTACT
ENDSCAN
CLOSE TABLES
```

9.2. Modularizarea programelor

Modularizarea programelor, în cazul unor aplicaţii complexe, se poate realiza prin proceduri şi funcţii definite de utilizator.

Definirea unei funcţii utilizator, reprezintă un bloc de comenzi independent, care primeşte un set de parametrii de la programul apelant şi returnează acestuia o valoare ca rezultat al prelucrării parametrilor de intrare.

Definirea unei proceduri de către utilizator, este de asemenea un bloc de comenzi, care prelucrează parametrii transmişi din programul apelant. După prelucrare controlul este redat programului apelant.

Din punct de vedere al locației, funcțiile se pot defini fie în programul apelant, fie într-o librărie sau pot fi salvate în fişiere externe, de tip .PRG.

9.2.1. Proceduri

Asocierea procedurilor cu programul apelant se face cu comanda:

SET PROCEDURE TO [<*fişier*_1> [,<*fişier*_2>,...]] [ADDITIVE]

unde:

- *fişier*_1, *fişier*_2 – specifică succesiunea în care vor fi deschise fişierele. Se pot declara mai multe nume de fişiere care conțin proceduri. Această opțiune permite crearea unor librării de sine stătătoare de funcții care pot fi specificate separat.
- ADDITIVE – se deschid fişierele cu proceduri din lista specificată, fără a fi închise fişierele cu proceduri, deja deschise.

Cu *comanda* RELEASE PROCEDURE [<*fişier*_1> [,<*fişier*_2>,...]] se închid fişierele individuale.

Variabilele definite în interiorul procedurilor şi funcțiilor utilizator, sunt la nivel local (nu se văd din afara procedurilor sau funcțiilor).

Structural o procedură, cuprinde următoarele elemente:

PROCEDURE *nume_procedură*	→	specifică numele procedurii create
PARAMETERS *listă_parametri*	→	declararea parametrilor formali de intrare
Bloc de comenzi	→	bloc de comenzi, corpul procedurii
RETURN [*expresie*]	→	ieşirea din procedură, opțional cu returnarea unei expresii calculate.

unde:

- PROCEDURE – este o declarație în interiorul unui program, care specifică începutul unei proceduri şi defineşte numele procedurii. Numele procedurii poate începe cu o literă sau

caracterul "_" (underscore) şi poate conține orice combinație de litere, numere şi caractere underscore. Lungimea maximă a numelui este de 254 caractere;

- PARAMETERS – asigură trecerea parametrilor din programul apelant în procedură. Se pot transmite maxim 27 de parametri în procedură. Lista parametrilor poate cuprinde: expresii, variabile de memorie, nume de tabele, nume de câmpuri din tabele de date, constante, care sunt transmise prin referință;
- RETURN [*expresie*] – returnează controlul şi opțional o expresie, în programul apelant;
- Parametrii pot fi transmişi în procedură, prin includerea clauzei PARAMETERS în procedură, sau prin plasarea unei liste de parametri imediat după PROCEDURE *nume_procedură*. Lista se inchide între paranteze, iar parametrii sunt separați cu virgulă.

Apelul unei proceduri sau program se face cu comanda

DO <*nume_procedură*> | <*nume_program*> WITH <*listă_parametri*>

Observații

- Metoda folosită la transmiterea parametrilor la proceduri sau funcții este implicit prin valoare, iar la programe prin referință;
- Variabilele transmise către procedură cu comanda DO...WITH sunt transmise prin referință.

Sistemul VFP are anumite *restricții legate de utilizarea comenzii DO*:

- pot fi imbricate maxim 32 de apeluri DO (*program principal* | *procedura*1 | *procedura*2 |...);
- nu se poate face apel dintr-o procedură la ea însăşi (nu admite recursivitatea directă);
- din interiorul unei proceduri nu poate fi apelat programul/procedura apelantă (nu admite recursivitatea indirectă).

9.2.2. Funcții

Elementele unei funcții definite de utilizator (UDF) sunt:

FUNCTION *nume_funcție*	→	specifică numele funcției create
Bloc de comenzi	→	bloc de comenzi, corpul funcției

RETURN [*expresie*] → ieşirea din funcție, opțional cu returnarea unei expresii calculate.

Implicit parametrii sunt transmişi în funcție prin valoare. Numărul maxim de parametri care poate fi transmis, este 27. Parametrii pot fi transmişi în funcție, prin includerea clauzei PARAMETERS în funcție, sau prin plasarea unei liste de parametri imediat după FUNCTION *nume_funcție*. Lista se inchide între paranteze, parametrii sunt separați cu virgulă.

Comanda

SET UDFPARAMS TO VALUE | REFERENCE

specifică dacă parametrii sunt transmişi către o funcție definită de utilizator (UDF) prin valoare sau prin referință.

Când o variabilă este transmisă prin valoare, valoarea variabilei poate fi modificată în funcția definită de utilizator, dar valoarea originală a variabilei în programul apelant nu se modifică.

Când o variabilă este transmisă prin referință şi funcția definită de utilizator modifică valoarea variabilei transmise, valoarea originală a variabilei în programul apelant de asemenea se modifică.

Implicit parametrii sunt transmişi prin valoare. Se poate forța transmiterea parametrilor către o funcție utilizator, în funcție de opțiunea aleasă în comanda SET UDFPARAMS. Variabilele se închid între paranteze pentru a forța transmiterea prin valoare. Forțarea transmiterii unui parametru prin referință se face punând în fața parametrului caracterul @.

Exemplul 1.

```
*** Transmiterea Variabilelor Prin Valoare ***
CLEAR
SET TALK OFF
WAIT 'APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN VALOARE'
WINDOW
SET UDFPARMS TO VALUE
STORE 1 TO GNX
*** Valoarea Lui Gnx Este Neschimbata ***
@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU(GNX))
@ 4,2 SAY 'VALOARE GNX: ' + STR(GNX)
*** TRANSMITEREA VARIABILEI PRIN REFERINTA***
```

```
WAIT ' APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN REFERINTA'
WINDOW
CLEAR
SET UDFPARMS TO REFERENCE
STORE 1 TO GNX
*** Valoarea Lui Gnx Se Schimba ***
@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU(GNX))
@ 4,2 SAY 'VALUE OF GNX: ' + STR(GNX)
SET UDFPARMS TO VALUE

*** Functie Utilizator (Udf) Care Aduna Unu La Un Numar ***
FUNCTION PLUSUNU
PARAMETER GNZ
GNZ = GNZ + 1
RETURN GNZ
```

Următorul exemplu arată cum sunt transmise variabilele prin valoare şi prin referinţă, utilizând parantezele şi caracterul @.

Exemplul 2.

```
*** Transmiterea Variabilei Prin Valoare ***
CLEAR
SET TALK OFF
WAIT ' APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN VALOARE '
WINDOW
STORE 1 TO GNX
@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU((GNX)))
@ 4,2 SAY 'VALOAREA GNX: ' + STR(GNX)
*** Transmiterea Variabilei Prin Referinta ***
WAIT ' APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN REFERINTA '
WINDOW
CLEAR
STORE 1 TO GNX
@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU(@GNX))
```

```
@ 4,2 SAY 'VALOARE GNX: ' + STR(GNX)

FUNCTION PLUSUNU(GNZ)
GNZ = GNZ + 1
RETURN GNZ
```

10. Comenzi ale nucleului SQL

Sistemul VFP suportă comenzile SQL care operează la nivel de tabelă (relație) şi o singură comandă SQL poate fi folosită pentru a înlocui o întreagă secvență de comenzi.

Sistemul VFP poate executa următoarele comenzi SQL:

1. *Comanda* SELECT – SQL

Comanda specifică criteriile pe baza cărora se bazează şi lansează interogarea (**Query**). Sistemul VFP interpretează interogarea şi extrage datele cerute din tabelă (tabele). Comanda SELECT pentru interogare poate fi lansată din următoarele zone:

- în fereastra de comenzi (modul de lucru interpretor);
- din interiorul unui program VFP;
- utilizând asistentul de proiectare pentru interogare (**Query Designer**).

Sintaxa comenzii:

SELECT [ALL | DISTINCT] [TOP *expr_num* [PERCENT]] *alias*.]*art_selectat* [AS *nume_coloană*] [[*alias*.]*art_selectat* [AS *nume_coloană*]…] ROM [FORCE] *nume_bază_de_date*!]*tabelă* [[AS] *alias_local*] [INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER] JOIN *nume_bază_de_date*!]*tabelă* [[AS] *alias_local*] [ON *condiție_join* [[INTO *destinație*] | [TO FILE *nume_fişier* [ADDITIVE] | TO PRINTER [PROMPT] | TO SCREEN]] [NOCONSOLE] [PLAIN] [WHERE *condiție_join* [AND *condiție_join* [AND | OR *condiție_filtrare* [AND | OR *condiție_filtrare*]] [GROUP BY *coloană_grup* [,*coloană_grup*] [HAVING *condiție_filtrare*] [UNION [ALL] *comandă_selectare*] [ORDER BY *articol_ordonat* [ASC | DESC] [,*articol_ordonat* [ASC | DESC]…]]

Semnificația clauzelor:

- SELECT – precizează câmpurile, constantele şi expresiile care vor fi afişate în urma interogării;
- ALL – implicit se vor afişa toate înregistrările din tabelă care îndeplinesc condițiile de selecție în rezultatul interogării;
- DISTINCT – se exclud duplicatele din înregistrările care îndeplinesc condițiile de selecție. Argumentul se poate folosi o singură dată în cadrul unei clauze SELECT;

- TOP *expr_num* [PERCENT] – rezultatul interogării va conține un număr specificat de înregistrări sau un anumit procent din înregistrări. Dacă se foloseşte clauza TOP trebuie inclusă şi clauza ORDER BY, care specifică coloana din care clauza TOP, va determina numărul de înregistrări (linii) care vor fi incluse în rezultatul interogării. Se pot specifica înregistrările în domeniul 1..32767. Înregistrările care au valori identice pentru coloanele specificate cu clauza ORDER BY sunt incluse în rezultatul interogării. De exemplu, dacă se specifică valoarea 10 pentru *expr_num*, rezultatul interogării poate conține mai mult de 10 înregistrări dacă există în tabelă mai mult de 10 linii care au valori identice în coloanele specificate cu clauza ORDER BY;
- *alias*. – se foloseşte pentru identificarea numelui câmpurilor (coloanelor) prin calificare. Fiecare câmp specificat cu *art_selectat* va genera o coloană în rezultatul interogării. Dacă două sau mai multe câmpuri au acelaşi nume, se va include alias-ul tabelei şi caracterul punct (.) în fața numelui câmpului, pentru a preveni duplicarea coloanei (calificare cu punct);
- *art_selectat* – specifică câmpul care trebuie inclus în rezultatul interogării. Acesta poate fi:
  - numele unui câmp din tabela care apare în clauza FROM;
  - constantă care specifică o valoare, care va apare în fiecare înregistrare din rezultatul interogării;
  - expresie care poate fi numele unei funcții utilizator;
- AS *nume_coloană* – specifică numele coloanei care va apare la ieşire, în rezultatul interogării. Această opțiune este utilă atunci când *art_selectat* este o expresie sau conține un câmp calculat şi vrem să dăm un nume sugestiv coloanei respective. Nu sunt admise decât caracterele care se folosesc la stabilirea numelui câmpurilor la proiectarea tabelelor (de exemplu nu se folosesc spații);
- FROM – se declară tabelele care conțin datele pe care vrem să le extragem în urma interogării. Dacă nu este deschis nici o tabelă, VFP va deschide caseta de dialog **Open** pentru a specifica locația fişierului care conține datele;
- FORCE – clauza prin care specifică că tabele sunt unite (join) în ordinea în care ele apar în clauza FROM;
- *nume_bază_de_date*![*tabelă*] – specifică numele unei baze de date, care nu este deschisă, care conține tabela cu date. Clauza se foloseşte în cazul în care tabela cu date nu face parte din baza de date curentă (deschisă). Semnul ! se foloseşte pentru adresarea tabelei prin calificare;

- [AS] *alias_local* – specifică un nume temporar pentru tabela menționată în argumentul tabelă;
- INNER JOIN – rezultatul interogării va conține numai înregistrările din tabelă care coincid cu una sau mai multe înregistrări, din altă tabelă (joncțiune internă);
- LEFT [OUTER] JOIN – rezultatul interogării va conține toate înregistrările din tabelă în stânga cuvântului JOIN şi înregistrările care coincid vor fi afişate la dreapta cuvântului JOIN. Clauza OUTER (exterior) este opțională, se include pentru a sublinia faptul că se crează o joncțiune exterioară;
- RIGHT [OUTER] JOIN – rezultatul interogării va conține toate înregistrările din tabelă la dreapta cuvântului JOIN şi înregistrările care coincid în stânga;
- FULL [OUTER] JOIN – rezulatul interogării va conține atât înregistrările care coincid, cât şi cele care nu coincid, din cele două tabele;
- [*nume_bază_de_date*!]*tabelă* [[AS] *alias_local*] – specifică numele tabelei (baza de date) cu care se realizează joncțiunea (eventual definit ca alias local);
- ON *condiție_join* – specifică condiția pentru care tabelele sunt unite (fac joncțiune);
- INTO *destinație* .- specifică locul unde vor fi stocate rezultatele interogării. Destinația poate fi una din următoarele clauze:
    - ARRAY *nume_tablou* – va stoca rezultatele într-o variabilă de memorie de tip matrice. Variabila nu se crează dacă rezultatul interogării furnizează 0 înregistrări;
    - CURSOR *nume_cursor* – stochează rezultatele interogării într-un cursor (fişier temporar). După executarea comenzii SELECT, cursorul temporar rămâne deschis (read-only) şi este activ. După închiderea cursorului temporar, fişierul este şters;
    - DBF | TABLE *nume_tabelă* – stochează rezultatele interogării într-o tabelă;
    - TO FILE *nume_fişier* – clauză prin care rezultatele interogării sunt direcționate către un fişier de tip ASCII (text);
    - ADDITIVE – adaugă rezultatele interogării la conținutul existent al fişierului de tip text specificat la clauza TO FILE;
    - TO PRINTER [PROMPT] – direcționează ieşirea către imprimantă;
    - TO SCREEN – direcționează ieşirea în fereastra principală a sistemului VFP sau în fereastra activă definită de utilizator;
    - NOCONSOLE – împiedică afişarea rezultatelor interogării trimise către un fişier, imprimantă sau fereastra principală;
- PLAIN – împiedică afişarea capului coloanei (numele coloanei) la ieşirea din interogare;

- WHERE - specifică includerea doar a anumitor înregistrări în rezultatele interogării;
- *condiție_join* [AND *condiție_join*…] – precizează câmpurile care leagă tabele din clauza FROM. Se include operatorul AND pentru a lega condiții multiple de unire (joncțiune);
- AND | OR *condiție_filtrare* [AND | OR *condiție_filtrare*…] – specifică criteriile pe care trebuie să le îndeplinească înregistrările pentru a fi incluse în rezultatul interogării. Se pot declara mai multe condiții de filtrare, legate prin operatorii AND şi/sau OR. Se poate folosi de asemenea şi operatorul NOT pentru a inversa valoarea expresiei logice;
- GROUP BY *coloană_grup* [,*coloană_grup*…] – grupează înregistrările pe baza valorilor din una sau mai multe coloane; *coloană_grup* poate fi numele unui câmp dintr-o tabela obişnuită, sau un câmp care include o funcție SQL, sau o expresie numerică care indică locația coloanei în tabela rezultată;
- HAVING *condiție_filtrare* – specifică o condiție de filtrare pe care grupul trebuie să o îndeplinească pentru a putea fi inclus în rezulatele interogării. HAVING trebuie utilizat împreună cu clauza GROUP BY. Clauza HAVING fără clauza GROUP BY acționează ca şi clauza WHERE;
- UNION [ALL] *comandă_selectare* - combină rezultatele finale ale unei clauze SELECT cu rezultatele finale ale altei clauze SELECT. Implicit UNION verifică rezultatele combinate şi elimină înregistrările duplicat. Clauza ALL împiedică acțiunea clauzei UNION de a elimina înregistrările duplicat.
- ORDER BY *articol_ordonat* ASC | DESC – sortează rezultatele interogării pe baza datelor din una sau mai multe coloane. Fiecare *articol_ordonat* trebuie să corespundă unei coloane din rezultatele interogării. ASC respectiv DESC specifică ordinea sortării (ascendent, descendent).

Exemplu. Din tabelele CLIENȚI şi COMENZI să se selecteze firma (CLIENȚI), data comenzii şi mijlocul de transport (COMENZI), sortate ascendent după dată. Rezultatul se va scrie în fişierul TRANSPORT.DBF.

```
OPEN DATABASES ('TEST')
SELECT A.FIRMA, B.DATA_CDA, B.MIJLOC FROM CLIENTI A, COMENZI B ;
WHERE A.COD_FIRMA=B.COD_FIRMA ;
ORDER BY B.DATA_CDA ASC INTO TABLE TRANSPORT
BROWSE
CLOSE DATABASES
```

2. *Comanda* ALTER TABLE – SQL modifică o tabelă existentă.

Sintaxa comenzii *:*

ALTER TABLE *nume_tabelă*_1 ADD | ALTER [COLUMN] *nume_câmp1*, *tip_câmp* [(*mărime_câmp* [,*precizie*])] [NULL | NOT NULL] [PRIMARY KEY | UNIQUE] [REFERENCES *nume_tabelă*_2 [TAG *nume_etichetă*]]

unde:

- *nume_tabelă*_1 – specifică numele tabelei a cărei structură se modifică;
- ADD [COLUMN] *nume_câmp*1 – specifică numele câmpului care se adaugă;
- ALTER [COLUMN] *nume_câmp*1 – specifică numele unui câmp existent care se modifică;
- *tip_câmp*[(*mărime_câmp*[,*precizie*])] – specifică tipul câmpului, mărimea şi precizia pentru un câmp nou sau pentru modificarea unui câmp existent;
- NULL | NOT NULL – permite declararea unui câmp care acceptă sau nu valori de tip **NUL;**
- PRIMARY KEY – crează o o etichetă primară de index. Eticheta de index are acelaşi nume cu cel al câmpului;
- UNIQUE – crează o etichetă de index candidat, cu acelaşi nume cu cel al câmpului;
- REFERENCES *nume_tabelă*_2 TAG *nume_etichetă* – specifică tabela părinte către care se stabileşte o relaţie persistentă. TAG *nume_etichetă* specifică eticheta de index din tabela părinte pe baza căreia se stabileşte relaţia.

3. *Comanda* UPDATE – SQL actualizează înregistrările dintr-o tabelă. Înregistrările pot fi actualizate pe baza rezultatelor unei declaraţii SELECT – SQL.

Sintaxa comenzii*:*

UPDATE [*nume_BD*!]*nume_tabelă* SET *nume_coloană=exprL*_1 [, *nume_coloană=exprL*_2…] WHERE *condiţie_filtrare*_1[AND | OR *condiţie_filtrare*_2…]]

unde:

- [*nume_BD*!]*nume_tabelă* – specifică tabela în care vor fi actualizate înregistrările, cu noile valori;

- SET *nume_coloană=exprL*_1[, *nume_coloană=exprL*_2…] – specifică coloanele care sunt actualizate şi noile valori. Dacă se omite clauza, fiecare înregistrare din coloană va fi actualizată cu aceaşi valoare;
- WHERE *condiție_filtrare*_1[AND | OR *condiție_filtrare*_2…]] – specifică înregistrările care vor fi actualizate cu noile valori; *condiție_filtrare*_1 specifică criteriul pe care trebuie să-l îndeplinească înregistrările pentru a fi actualizate. Se pot include mai multe condiții de filtrare legate prin operatorii logici AND şi/sau OR. Se poate folosi de asemenea operatorul NOT pentru a inversa valoarea expresiei logice.

Exemplu. În fişierul CLIENȚI din baza de date TEST, să se modifice câmpul cantitate maximă (CANT_MAX) la valoarea 25.

```
OPEN DATABASES ('TEST')
USE CLIENTI
UPDATE CLIENTI SET CANT_MAX=25
BROWSE FIELDS FIRMA,CANT_MAX
CLOSE DATABASES
```

4. *Comanda* INSERT – SQL adaugă o nouă înregistrare la sfârşitul unei tabele existente. Noua înregistrare conține date descrise în comanda INSERT sau pot fi preluate dintr-un masiv.

Sintaxa comenzii*:*

INSERT INTO *tabelă* [(*nume_câmp*_1[, *nume_câmp*_2,...])] VALUES (*valoare*_1, *valoare*_2,...])

sau

INSERT INTO t*abelă* FROM ARRAY *nume_masiv* | FROM MEMVAR

unde:

- *tabelă* – specifică numele tabelei în care se adaugă o nouă înregistrare;
- (*nume_câmp*_1[, *nume_câmp*_2,...]) – specifică numele câmpurilor din din noua înregistrare în care vor fi inserate valori;
- (va*loare*_1, *valoare*_2,...]) – specifică valorile câmpurilor care vor fi inserate în noua înregistrare;
- *nume_masiv* – specifică numele masivului din care vor fi inserate datele în noua înregistrare;

- FROM MEMVAR – conținutul variabilelor va fi inserat în câmpurile care au acelaşi nume cu variabilele.

Exemplu. În tabela CLIENȚI din baza de date TEST conținutul înregistrării curente va fi transmis în memorie ca variabile şi structura tabelei va fi copiată într-un noua tabelă CLIENȚI2.

```
CLOSE DATABASES
CLEAR
OPEN DATABASES ('TEST')
USE CLIENTI
* Se transmite înregistrarea curentă în memorie ca variabile
SCATTER MEMVAR
* Se copiază structura tabelei curente în tabela CLIENTI2
COPY STRUCTURE EXTENDED TO CLIENTI2
* Se inserează înregistrarea memorată în variabile
INSERT INTO CLIENTI2 FROM MEMVAR
SELECT CLIENTI2
BROWSE
USE
DELETE FILE CLIENTI2.DBF
```

5. *Comanda* CREATE CURSOR – SQL crează o tabelă temporară. Fiecare câmp din tabela temporară este definit cu nume, tip, precizie, număr zecimale, valoare **NULL** şi reguli de integritate referențială. Aceste definiții pot fi obținute din comandă sau dintr-un masiv.

Sintaxa comenzii:
CREATE CURSOR *nume_alias*(*nume_câmp*_1 *tip_câmp* [*precizie*[,*nr_zecimale*])
[NULL | NOT NULL] [CHECK *expr_L* [ERROR *mesaj_eroare*]] [DEFAULT *expresie*]
[UNIQUE] [NOCPTRANS]] [, *nume_câmp*_2...]) | FROM ARRAY *nume_masiv*

unde:

- *nume_alias* – specifică numele tabelei temporare creată, care poate fi şi numele unei expresii;
- *nume_câmp*_1 – specifică numele unui câmp din fişierul temporar;

- *tip_câmp* – prin intermediul unei singure litere, specifică tipul fiecărui câmp;
- *precizie*[,*nr_zecimale*] – specifică mărimea şi dacă este cazul, numărul de zecimale pentru câmpuri numerice;
- NULL | NOT NULL – alocă sau nu valori de tip **NULL** câmpului;
- CHECK *expr_L* – specifică regula de validare pentru valorile care se vor înscrie în câmp. Expresia logică *expr_L* poate fi şi o funcţie definită de utilizator;
- ERROR *mesaj_eroare* – specifică mesajul de eroare pe care sistemul VFP îl va afişa în cazul în care validarea datelor generează eroare;
- DEFAULT *expresie* – specifică valoarea implicită pentru câmp. Tipul de dată dat de expresie trebuie să fie de acelaşi fel cu tipul câmpului.
- UNIQUE – crează un index candidat pentru câmp. Eticheta de index candidat are acelaşi nume cu cel al câmpului. Valorile **NULL** şi înregistrările duplicat nu sunt permise într-un câmp utilizat ca index candidat;
- NOCPTRANS – previne trecerea la o altă pagină de cod pentru caractere şi câmpuri de tip memo. Dacă cursorul este convertit la o altă pagină de cod, câmpurile pentru care a fost specificată clauza NOCPTRANS nu vor fi translatate;
- FROM ARRAY *nume_masiv* – specifică numele unui masiv existent care conţine numele, tipul, precizia, numărul de zecimale pentru fiecare câmp din tabela temporară.

Exemplu. Se crează un cursor cu nume alias ANGAJAŢI cu următoarea structură:
ID N(5) – identificator angajat;
NUME C(25);
ADRESA C(30);
ORAS C(20);
TELEFON C(8) – care acceptă şi valori de tip **Null**;
SPEC M – specialitatea, câmp de tip memo.

Se va adăuga o înregistrare goală, după care se vor înscrie valori în câmpurile tabelei.

```
CLOSE DATABASES
CLEAR
CREATE CURSOR ANGAJATI ;
(ID N(5), NUME C(25), ADRESA C(30), ORAS C(20), TELEFON C(8) NULL, SPEC M)
DISPLAY STRUCTURE
```

WAIT WINDOW ‘Apasati o tasta pentru a introduce o inregistrare’
INSERT INTO ANGAJATI (ID, NUME, ADRESA, ORAS, MARCA, SPECIALITATE);
VALUES (1004,”DR. ION GARCEA”,”B-DUL LACUL TEI 124, SECTOR 2”, ”BUCURESTI”,;
“88567902”,”PAZA SI PROTOCOL”)
BROWSE
* În acest punct se poate copia înregistrarea într-o tabelă permanentă.
CLOSE ALL && odată ce cursorul s-a închis, toate datele sunt golite din memorie.
CLEAR

6. *Comanda* CREATE TABLE – SQL determină crearea unei tabele. La fiecare tabelă nou creată, i se specifică numele câmpurilor din înregistrare şi caracteristicile lor: tip, mărime, zecimale (pentru tipurile numerice), valori de tip **NULL** şi regulile de integritate referențiale. Definirea câmpurilor se poate obține fie prin descriere în comandă, fie dintr-un masiv.

Sintaxa comenzii:
CREATE TABLE | DBF *nume_tabelă*_1 [NAME *nume_lung*] [FREE] (*nume_câmp*_1 *tip_câmp* [*mărime*[(,*mărime_câmp*[,*precizie*])] [NULL | NOT NULL] [CHECK *expr_L*_1 [ERROR mesaj_eroare_1]] [DEFAULT *expr*_1] [PRIMARY KEY | UNIQUE] [REFERENCES *nume_tabelă*_2 [TAG *etich*_1_*index*]] [NOCPTRANS] [*nume_câmp*_2 …]
[PRIMARY KEY *expr*_2 TAG *etich*_2_*index*|, UNIQUE *expr*_3 TAG *etich*_3_*index*]
[, FOREIGN KEY *expr*_4 TAG *etich*_4_*index* [NODUP] REFERENCES *nume_tabelă*_3 [TAG *etich*_5_*index*]]
[,CHECK *expr_L*_2[ERROR *mesaj_eroare*_2]]) | FROM ARRAY *nume_masiv*

unde:

- *nume_tabelă*_1 – specifică numele tabelei care va fi creată. Opțiunile TABLE şi DBF sunt identice;
- NAME *nume_lung* – specifică un nume lung pentru tabelă. Acest nume poate fi specificat numai dacă este deschisă o bază de date, el fiind stocat (memorat) în cadrul bazei de date. Numele lung poate conține până la 128 caractere şi poate fi folosit numai în cadrul bazei de date;
- FREE – noua tabelă nu va fi adăugată la baza de date curentă (deschisă);
- (*nume_câmp*_1 *tip_câmp* [*mărime*[(,*mărime_câmp*[,*precizie*])] – specifică numele câmpului, tipul, mărimea şi precizia (numărul de poziții zecimale). O tabelă poate conține până la 255

câmpuri. Dacă unul sau mai multe câmpuri acceptă valori de tip **NULL**, limita se reduce la 254;

- NULL | NOT NULL – permite / împiedică introducerea de valori de tip **NULL** în câmp;
- CHECK *expr_L*_1 – specifică o regulă de validare pentru câmp. Expresia logică *expr_L*_1 poate fi o funcție definită de utilizator. La adăugarea unei înregistrări vide, se verifică regula de validare. Dacă regula de validare nu prevede acceptarea de valori vide în câmp, se generează eroare;
- ERROR *mesaj_eroare*_1 – specifică mesajul de eroare pe care sistemul VFP îl afişează la apariția unei erori, generate de clauza CHECK, care verifică regula de validare;
- DEFAULT *expr*_1 – specifică valoarea implicită pentru câmp. Expresia *expr*_1 trebuie să fie de acelaşi tip cu tipul câmpului;
- [PRIMARY KEY – crează un index primar pentru câmp. Eticheta de index primar are acelaşi nume cu cel al câmpului;
- UNIQUE – crează un index candidat pentru câmp. Numele etichetei de index candidat este acelaşi cu cel al câmpului.
- REFERENCES *nume_tabelă*_2 [TAG *etich*_1_*index*] – specifică numele tabelei părinte, la stabilirea unei relații persistente. Dacă se omite clauza TAG *etich*_1_*index*, relația se stabileşte utilizând cheia primară de index a tabelei părinte. Dacă tabela părinte nu are un index de cheie primară, sistemul va genera eroare. Se include clauza TAG *etich*_1_*index* pentru a stabili o relație bazată pe existența etichetei de index pentru tabela părinte;
- NOCPTRANS – previne trecerea la o altă pagină de cod pentru caractere şi câmpuri memo. Dacă tabela este convertită la o altă pagină de cod, câmpul pentru care a fost specificată clauza NOCPTRANS, nu va fi translatat;
- [nume_câmp_2 …] – următorul câmp din structura tabelei. Are aceleaşi caracteristici de descriere ca şi primul câmp;
- PRIMARY KEY *expr*_2 TAG *etich*_2_*index* – specifică indexul primar care va fi creat. Expresia *expr*_2 poate specifica orice câmp sau combinație de câmpuri din tabelă. TAG *etich*_2_*index* specifică numele etichetei de index primar. Deoarece o tabelă nu poate avea decât o singură cheie primară de indexare, nu se poate include această clauză dacă deja există definită o cheie primară;
- UNIQUE *expr*_3 TAG *etich*_3_*index* – crează un index candidat. Expresia *expr*_3 specifică orice câmp sau combinație de câmpuri din tabelă. Clauza TAG *etich*_3_*index* specifică numele etichetei de index pentru eticheta de index candidat, care va fi creat;

- FOREIGN KEY *expr*_4 TAG *etich*_4_*index* [NODUP] – crează un index extern (non-primar) şi stabileşte o relaţie cu tabela părinte. *expr*_4 specifică expresia indexului extern. Clauza TAG *etich*_4_*index* specifică numele etichetei de index externe. Clauza NODUP se include pentru a crea un index candidat extern;
- REFERENCES *nume_tabelă*_3 [TAG *etich*_5_*index*] – specifică tabela părinte către care se stabileşte o relaţie persistentă. Includerea clauzei TAG *etich*_5_*index* determină stabilirea relaţiei pe baza unei etichete de index a tabelei părinte. Dacă este omisă clauza, relaţia se stabileşte utilizând implicit cheia primară de indexare din tabela părinte;
- CHECK *expr_L*_2[ERROR *mesaj_eroare*_2] – specifică regula de validare pentru tabelă;
- FROM ARRAY *nume_masiv* – specifică numele unui masiv existent care conţine numele, tipul, mărimea şi precizia pentru fiecare câmp al tabelei. Conţinutul masivului poate fi definit cu funcţia AFIELDS().

7. *Comnda* DELETE – SQL realizează ştergerea la nivel logic (marcarea pentru ştergere) a înregistrărilor dintr-o tabelă.

Sintaxa comenzii:

DELETE FROM [*nume_BD*!]*nume_tabelă* [WHERE *condiţie_filtrare*_1 [AND | OR *condiţie_filtrare*_2...]]

unde:

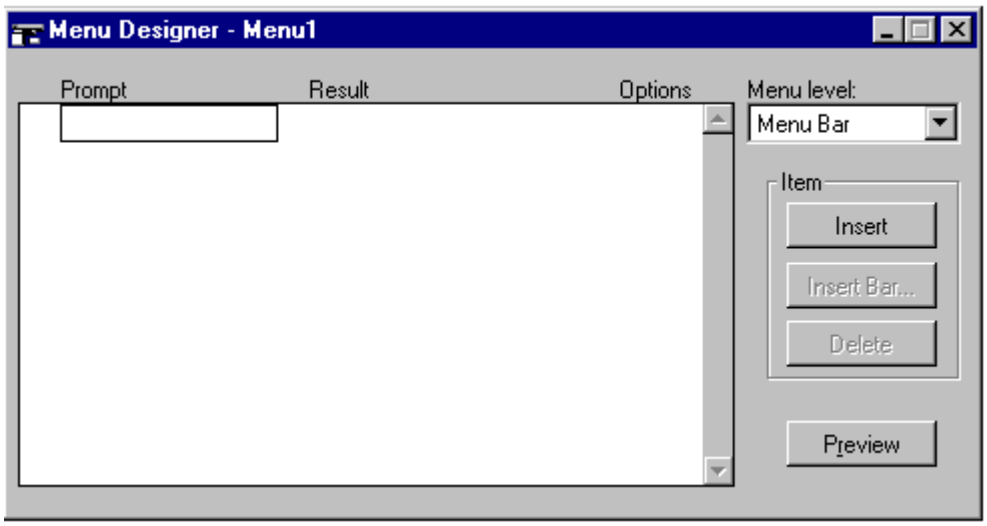
- FROM [*nume_BD*!]*nume_tabelă* – specifică numele tabelei în care înregistrările sunt marcate pentru ştergere la nivel logic;
- WHERE *condiţie_filtrare*_1 [AND | OR *condiţie_filtrare*_2...] – vor fi marcate pentru ştergere numai anumite înregistrări, cele care îndeplinesc condiţiile de filtrare.

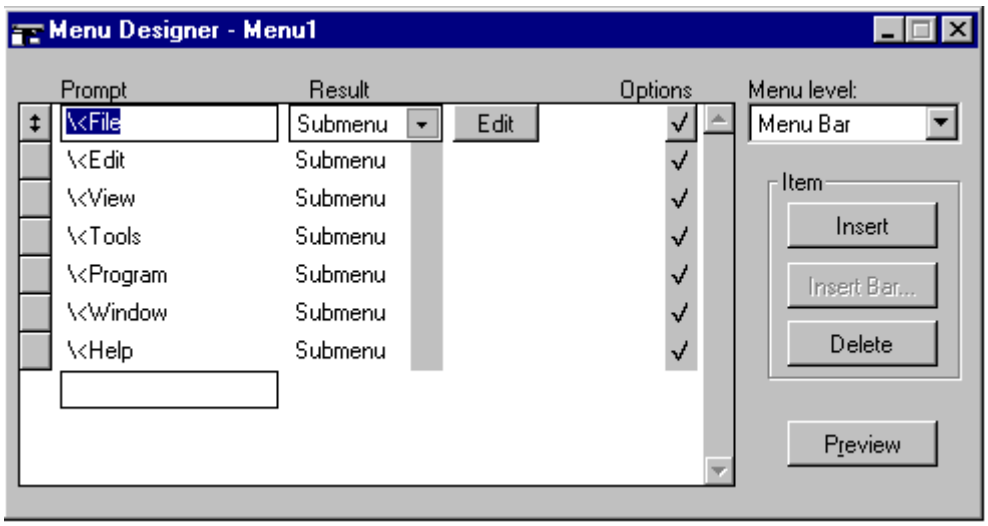
Ştergerea fiind la nivel logic, înregistrările vor fi şterse fizic din tabelă doar după utilizarea comenzii PACK.

11. Proiectarea meniurilor şi a barelor de instrumente

Meniurile (**Menus**) şi barele de instrumente (**Toolbars**) furnizează o cale structurată şi accesibilă pentru mânuirea comenzilor aplicației. Prin planificarea şi proiectarea judicioasă a meniurilor şi barelor de instrumente, se poate creşte calitatea aplicațiilor.

O bună parte din activitatea de creare a unui meniu sistem este realizată cu ajutorul proiectantului de meniuri (**Meniu Designer**), în care se crează meniul principal, submeniurile şi opțiunile din meniu.





12. Aplicații

12.1. Evidența rezultatelor activității studenților într-o facultate

12.1.1. Formularea şi analiza problemei

Enunțul problemei

Tema abordată este *Evidența rezultatelor activității studenților într-o facultate.*

Datele de intrare

Fişa de înscriere;

Cataloagele de note;

Planul de învățământ;

Programele analitice.

Datele de ieşire

Componența grupelor;

Situația după o sesiune normanală;

Situația statistică asupra repartiției notelor la o disciplină;

Situația statistică asupra repartiției mediilor.

Actualizarea datelor

Adăugarea de noi studenți;

Modificarea datelor despre studenți;

Ştergerea unui student;

Adăugrea de note;

Modificarea notelor;

Ştergerea studentului din fişierul de note;

Adugarea de noi discipline;

Modificarea numelui unei discipline;

Ştergerea unei discipline.

Coduri adoptate

Codul județului;

Codurile disciplinelor;

An studii 1-6, an calendaristic 2006-2007.

Interfața cu utilizatorul

Stabilirea meniurilor şi a videoformatelor pentru programul monitor şi a programelor subordonate.

Proiectarea intrărilor

Cererea de înscriere;

Cataloage;

Formularele cu codurile şi denumirile disciplinelor.

Proiectarea ieşirilor

Lista studenților pe grupe;

Situația după sesiune;

Statistica la o disciplină;

Statistica medii;

Listă bursieri;

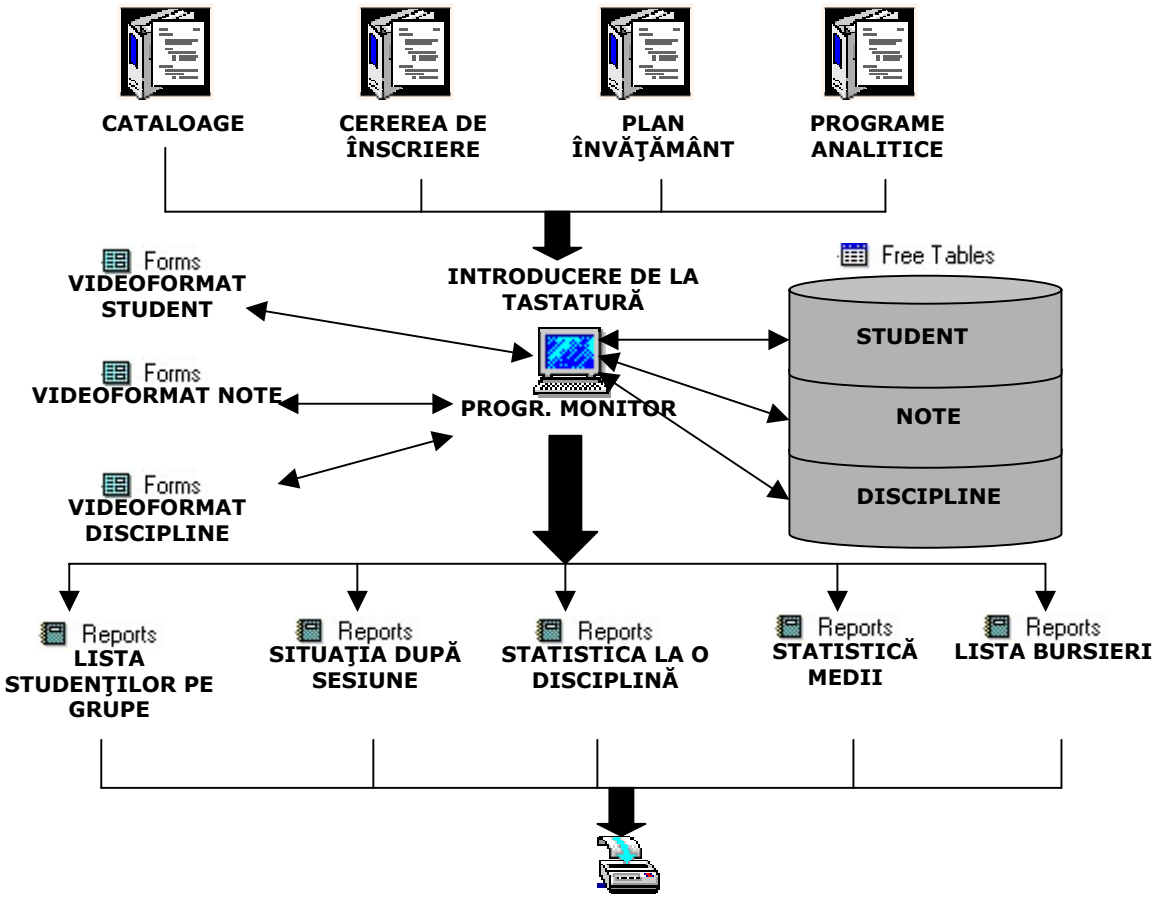
Listă integralişti.

Stabilirea cadrului pentru:

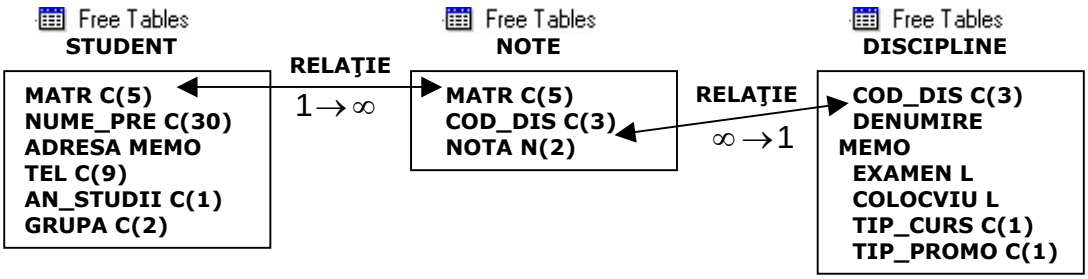
Operațiile de actualizare

Obținerea rapoartelor finale.

Schema generală propusă



12.1.2. Crearea, actualizarea, modificarea şi interogarea tabelelor

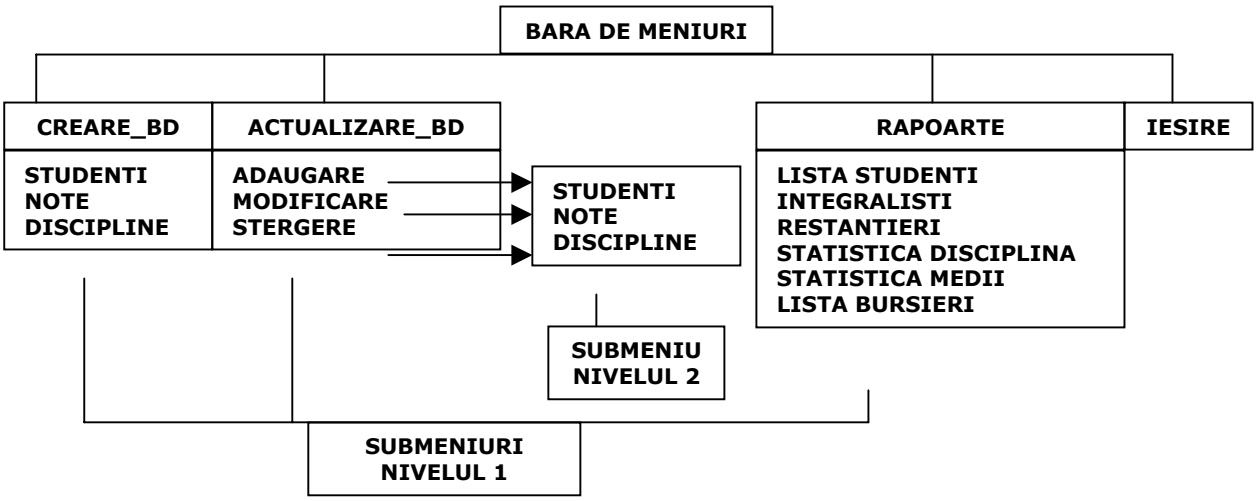


CODIFICARI

TIP_CURS
- O - OBLIGATORIU
- F - FACULTATIV
- P - OPTIONAL

TIP_PROMO
- O - OBLIGATORIU
- N - Nu este obligatoriu pentru promovare

Meniul programului ar putea fi de forma



12.1.3. Crearea structurii unui fişier

În fereastra de comenzi se introduc comenzile de creare a unui fişier şi vizualizare:

CREATE TABLE STUD1 (MATR C(3), NUME C(30), AN C(1), GRUPA C(2))

BROWSE

Se lansează comenzile **[CTRL]** + **[Y]** şi se populează cu 3-4 înregistrări. Se salvează cu **[CTRL]** + **[W]**.

Se deschide fereastra **Data Session** şi se închide fişierul STUD1 de la butonul **Close**.

Se va crea programul de modificare a structurii fişierului anterior pentru adăugarea unui nou câmp cu numele *mbac* (medie bacalaureat): **File**/**New**/**Program**/**New File** şi introduc următoarele linii:

```
SELECT 2
USE STUD1
COPY STRUCTURE EXTENDED TO TEMP
USE TEMP
BROWSE
APPEND BLANK
REPLACE FIELD_NAME WITH 'MBAC'
REPLACE FIELD_TYPE WITH 'N'
REPLACE FIELD_LEN WITH 5
REPLACE FIELD_DEC WITH 2
```

```
BROWSE
USE TEMP
CREATE STUD2 FROM TEMP
USE STUD2
APPEND FROM STUD1
BROWSE
CLOSE DATABASES
```

Se salvează şi se rulează programul.

12.1.4. Adresarea prin macrosubstituție

Se va crea un program care va folosi macrosubstituția pentru numele de fişiere STUD1 şi STUD2. Se vor introduce următoarele linii de program:

```
PUBLIC FIS C(20)
FOR I=1 TO 2 DO
    GOTO I
    FIS='STUD'+ALLTRIM(STR(I))+'.DBF'
    USE &FIS
    BROWSE
ENDFOR
CLOSE DATABASES
```

Se salvează şi se rulează programul.

12.1.5. Crearea unui meniu

Menu Bar va avea următoarele componente:

Creare BD – **Results** – **Submenu** – **Create** – Studenti/Note – **Results - Procedure**

Actualizare - **Results** – **Submenu** – **Create** – Studenti/Note – **Results - Procedure**

Media – **Mesults- Procedure**

Iesire– **Results- Procedure**

a) Procedurile pentru crearea BD

Pentru fişierul *Studenti*

```
CREATE TABLE STUD (MATR C(3), NUME C(30), AN C(1), GRUPA C(2))
CLOSE DATABASES
```

Pentru fişierul *Note*

```
CREATE TABLE NOTE (NOTA1 N(2), NOTA2 N(2), MEDIA N(5,2))
CLOSE DATABASES
```

b) Procedurile de actualizare

Pentru fişierul *Studenti*

```
USE STUD
BROWSE
CLOSE DATABASES
```

Pentru fişierul *Note*

```
USE NOTE
BROWSE
CLOSE DATABASES
```

În fiecare fereastră **Browse** cu **[CTRL]** + **[Y]** se populează fişierele cu 3-4 înregistrări. La fişierul *Note* nu se completează câmpul *media*.

c) Procedura de calcul a mediei

```
USE NOTE
FOR I=1 TO RECCOUNT()
    GOTO I
    REPLACE MEDIA WITH (NOTA1+NOTA2)/2
ENDFOR
```

```
BROWSE
CLOSE DATABASES
```

d) Procedura de ieşire

```
CLEAR EVENTS
CLOSE DATABASES
SET SYSMENU TO DEFAULT
```

12.2. Gestionarea unei magazii

Să se realizeze o aplicaţie în Visual Fox Pro 6 prin care să se gestioneze intrările de materiale, furnizorii şi materialele într-o magazie a unei secţii de producţie.

12.2.1. Proiectarea şi realizarea aplicaţiei

Aplicaţia va cuprinde un modul pentru definirea bazei de date, un modul pentru actualizare (adăugare, modificare, ştergere) şi un modul de exploatare a bazei de date prin rapoarte.

Sunt prevăzute trei tabele independente care au următoarea structură:

Fişierul *materiale.dbf*		Fişierul *furnizori.dbf*		Fişierul *intrari.dbf*	
Denumire	**Tip**	**Denumire**	**Tip**	**Denumire**	**Tip**
Denumire	C(40)	Denumire	C(40)	Cod_mat	N(5)
Cod_mat	N(5)	Cod_furn	N(4)	Cod_furn	N(4)
Um	C(6)	Adresa	M	Cant	N(6)
		Tel	C(10)	Pu	N(8,2)
				Data_in	D

Existenţa câmpurilor comune din fişierul *intrari.dbf* cu cele ale celorlalte fişiere oferă posibilitatea ca în etapa de proiectare a rapoartelor să se poată stabili relaţii între tabele pentru a extrage date corelate din toate sursele.

Se vor realiza două rapoarte:

- pentru obținerea unei liste de intrări de materiale sortată ascendent după data calendaristică cu câteva comenzi SQL de manipulare a tabelelor;
- pentru obținerea unei liste cu furnizori şi materiale folosind facilitatea de grupare a modulului de rapoarte. Se vor utiliza date extrase din toate cele trei fişiere (stabilind relații între acestea), calculându-se valoarea plătită fiecărui furnizor.

Aplicația conține proceduri pentru întreținerea şi exploatarea bazei de date fiind construită cu o interfață accesibilă şi logică. Structura proiectului este prezentată în Figura 1.

Se va utiliza programare vizuală, orientată spre obiecte şi procedurală în realizarea aplicației care rezolvă problema formulată mai sus.

În programul principal *init_var* sunt declarate şi inițializate variabilele globale, se deschid sau se crează fişierele utilizate şi se lansează meniul aplicației.

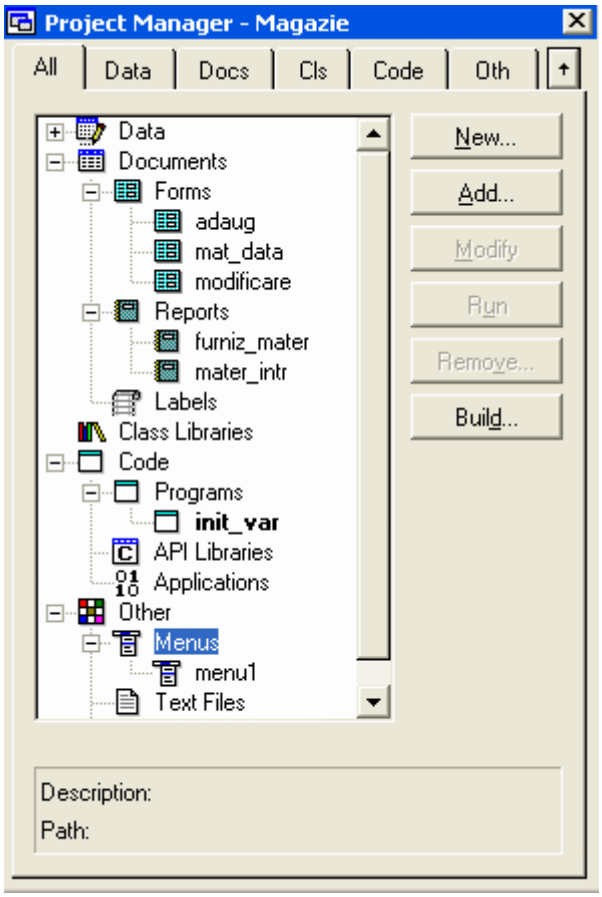


Fig. 1

Programul principal – *Init_var* – din secțiunea **Programs**

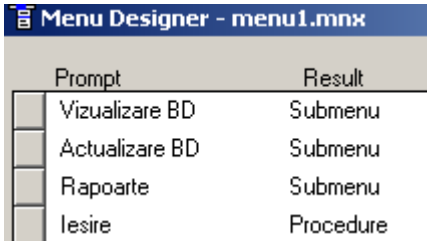
PUBLIC CODF, DEN, ADR, TL, CODM, DENM, PUN, DATAIN, CANTIT

```
SET DATE TO BRITISHCODF=0
*  INITIALIZARI
DEN=' '
ADR=' '
TL=' '
CODM=0
DENM=' '
PUN=0
DATAIN={  /  /  }
CANTIT=0
CLOSE DATABASES
IF EMPTY(SYS(2000,'MATERIALE.DBF'))
      CREATE TABLE MATERIALE (COD_MAT N(5),DENUMIRE C(40),UM C(6))
ELSE
      USE MATERIALE
ENDIF

INDEX ON COD_MAT TAG COD_MAT ADDITIVE
IF EMPTY(SYS(2000,'INTRARI.DBF'))
      CREATE TABLE INTRARI (COD_MAT N(5), COD_FURN N(4), CANT N(6), PU N(8,2),
DATA_IN D)
ELSE
      USE INTRARI
ENDIF

IF EMPTY(SYS(2000,'FURNIZORI.DBF'))
      CREATE TABLE FURNIZORI (COD_FURN N(4), DENUMIRE C(40), ADRESA M, TEL
C(10))
ELSE
      USE FURNIZORI
ENDIF
INDEX ON COD_FURN TAG COD_FURN ADDITIVE

CLOSE DATABASE
```

SELECT 2
USE MATERIALE
SELECT 3
USE FURNIZORI
SELECT 4
USE INTRARI
DO MENU1.MPR
READ EVENTS

În continuare se defineşte bara de meniuri. În Figura 2 sunt prezentate *Meniul principal* şi submeniul *Vizualizare BD*.

Meniul principal



Submeniu Vizualizare BD

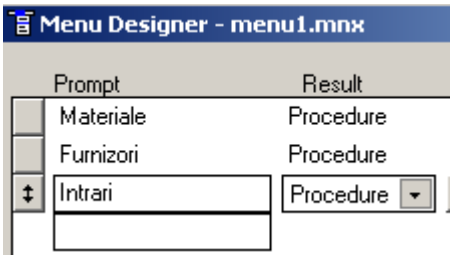


Fig. 2

***Crearea submeniului* Vizualizare BD**

* **Vizualizarea tabelei** *Materiale*

SELECT 2
BROWSE FIELDS COD_MAT, DENUMIRE, PU, DATA_IN NOEDIT

* **Vizualizarea tabelei** *Furnizori*

SELECT 3
BROWSE FIELDS COD_FURN, DENUMIRE NOEDIT

* **Vizualiarea tabelei** *Intrari*

SELECT 4
BROWSE NOEDIT

Crearea submeniu *Actualizare BD*

Submeniul este prezentat în Figura 3.

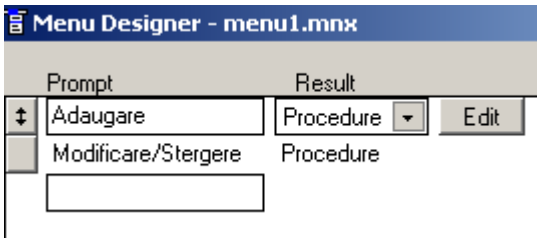


Fig. 3

Opțiunea *Adaugare*

DO FORM ADAUG

Opțiunea *Modificare/Stergere*

DO FORM MODIFICARE

* **Crearea submeniu** *Rapoarte*

Submeniul este prezentat în Figura 4:

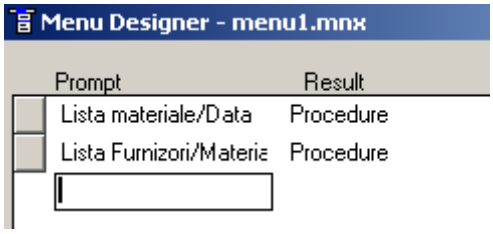


Fig. 4

Opțiunea *Lista materiale/Data*

DO FORM MAT_DATA

Opțiunea *Lista Furnizori/Materiale*

Se declară următoarea procedură:

```
SELECT 4
SET ORDER TO TAG COD_MAT OF MATERIALE.CDX IN MATERIALE
SET RELATION TO COD_MAT INTO MATERIALE ADDITIVE
```

```
SET ORDER TO TAG COD_FURN OF FURNIZORI.CDX IN FURNIZORI
SET RELATION TO COD_FURN INTO FURNIZORI ADDITIVE
REPORT FORM FURNIZ_MATER PREVIEW
SET RELATION OFF INTO MATERIALE
SET RELATION OFF INTO FURNIZORI
```

care stabileşte o relaţie între înregistrările fişierului *materiale.dbf* şi *intrari.dbf* prin intermediul câmpului comun *cod_mat* şi pe de altă parte o relaţie între înregistrările fişierului *furnizori.dbf* şi *intrari.dbf* prin intermediul câmpului comun *cod_furn*.

Pentru stabilirea relaţiilor între fişiere trebuie ca în prealabil fişierele *copil* să fie indexate după câmpul de legătură (*cod_mat* în fişierul *materiale, cod_furn* în fişierul *furnizori*), etapă realizată în programul principal *init_var*.

După vizualizarea raportului *furniz_mater* cele două relaţii sunt anulate.

Opţiunea *Iesire*

```
CLEAR EVENTS
CLOSE DATABASE
SET SYSMENU TO DEFAULT
```

* **Videoformat** *Adaug*

Aceast videoformat are numele intern **Form1**, numele extern *Adaug.scx,* iar pentru proprietatea **Caption** se alege textul *Adaugare*, Figurile 5a şi 5b.

Butonul *Iesire* este primul strat şi se aplica deasupra obiectului **PageFrame1** (de tip container) având proprietatea **PageCount** =3 (pagini). Pe butonul Iesire la evenimentul **Click** se asociaza comanda *THISFORM.RELEASE*.

Se face click cu butonul din dreapta al mouse-ului şi se alege opţiunea **Edit** pentru a accesarea elementelor containerului, respectiv cele trei pagini: *Furnizori*, *Materiale*, *Intrari*. Numele extern şi grosimea de caracter a paginilor se stabileşte utilizând proprietăţile **Caption** şi **FontBold**= **True**.

Adresarea paginilor se realizează făcând click cu butonul stâng al mouse-ului pe fiecare pagină. Numele intern poate fi schimbat, dar vor fi păstrate cele implicite (**Page1, Page2, Page3**), respectiv adresarea paginii *Furnizori* în procedurile ataşate va fi una ierarhică, prin calificare:

THISFORM.PAGEFRAME1.PAGE1.*proprietate*

Tabela *Furnizori*

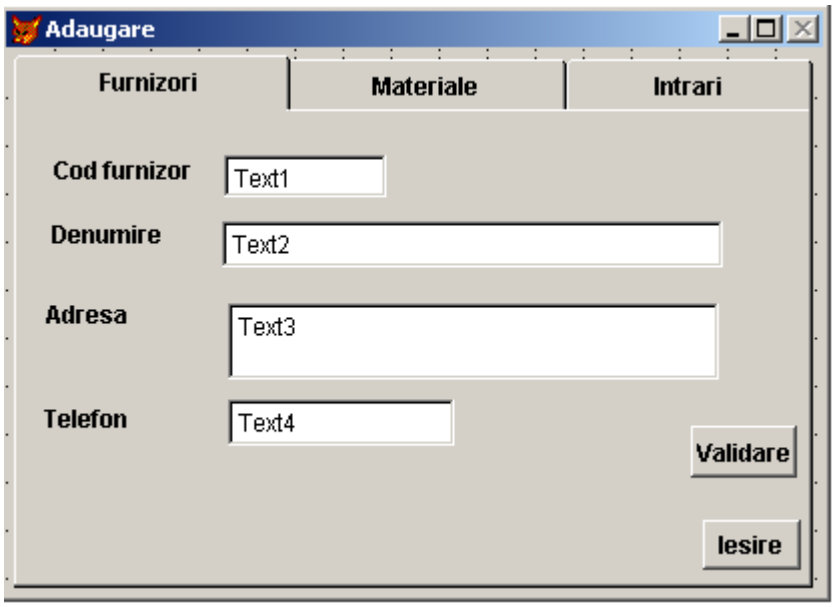


Fig. 5a

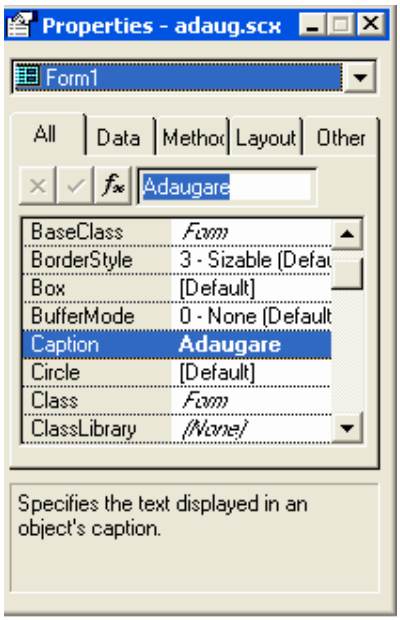


Fig.5b

Urmează stabilirea obiectelor pentru pagina **Page1** (cu proprietatea **caption**='*Furnizori*'). Sunt patru regiuni de tip text, notate de la **Text1** la **Text4,** un buton de comandă (**Command1**) şi patru etichete, de la **Label1** la **Label4**. Ordinea lor se stabileşte din meniul **View**, submeniul **TabOrder**. Dacă se doreşte trecerea de la vizualizarea interactivă a ordinii obiectelor la vizualizarea de tip listă, se selectează meniul **Tools,** submeniul **Options**, submeniul **Forms**, opțiunea **Tab Ordering** şi se alege **By list**.

În continuare se stabilesc câteva proprietăți referitoare la obiectele alese:

- pentru obiectul **Page1** (*Furnizori*) se accesează codul procedurilor ataşate diferitelor evenimente utilizând butonul dreapta al mouse-ului şi alegând din meniu opțiunea **Code**. Se selectează evenimentul **Activate** şi se adăugă următoarele linii de cod în editorul deschis:

```
* SE SELECTEZĂ ZONA DE LUCRU 3 (FURNIZORI.DBF)
SELECT 3
* SE TRIMITE FOCUSUL (PROMPTERUL) PE TEXT1
THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS
CODF=0
DEN ="
```

```
ADR=' '
TL=''
* ACTUALIZAREA REGIUNILOR DE EDITARE CU NOILE VALORI
THISFORM.PAGEFRAME1.PAGE1.REFRESH
```

- pentru obiectele de tip text la proprietăți, secțiunea **Data**, se declară variabilele în **ControlSource**, care vor prelua datele ce se vor scrie în fişier: **Text1-** *codf,* **Text2-** *den***, Text3-** *adr***, Text4-** *tl*.
- Etichetele vor avea trecute la proprietatea **Caption** denumirile explicite a variabilelor utilizate: *Cod furnizor (***Label1***), Denumire (***Label2***), Adresa* **(Label3***), Telefon (***Label4***)*. Proprietatea **FontBold** se setează **True**.
- Butonul de validare (nume intern **Command1**, proprietatea **Caption=Validare**, **FontBold=True**) va avea ataşate următoare linii de cod pentru evenimentul **Click**:

```
LOCATE FOR CODF=COD_FURN
IF FOUND(3) THEN
    WAIT WINDOW 'COD FURNIZOR DUPLICAT !'
ELSE
    LOCATE FOR UPPER(ALLTRIM(DEN))=UPPER(ALLTRIM(DENUMIRE))
    IF FOUND(3) THEN
        WAIT WINDOW 'DENUMIRE DUPLICAT !'
    ELSE
        APPEND BLANK
        REPLACE COD_FURN WITH CODF, DENUMIRE WITH DEN, ADRESA WITH
ADR, TEL WITH TL
    ENDIF
ENDIF
CODF=0
DEN=' '
ADR=' '
TL=' '
THISFORM.PAGEFRAME1.PAGE1.REFRESH
THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS
```

Secvența de cod realizează două validări, una pentru codul de furnizor şi una pentru denumirea firmei, verificând dacă mai există o înregistrare similară. Verificarea se face pentru a păstra caracterul de unicitate al codului şi denumirii. În caz ca se găseşte un duplicat se afişează un mesaj specific. Dacă codul şi denumirea sunt valide din punct de vedere al unicității, se trece la scrierea în fişierul de *furnizor.dbf* a noilor informații. Se reiniționalizează variabilele, se actualizează câmpurile afişate şi se trimite focusul (prompter-ul) pe obiectul **Text1.**

Tabela *Materiale*

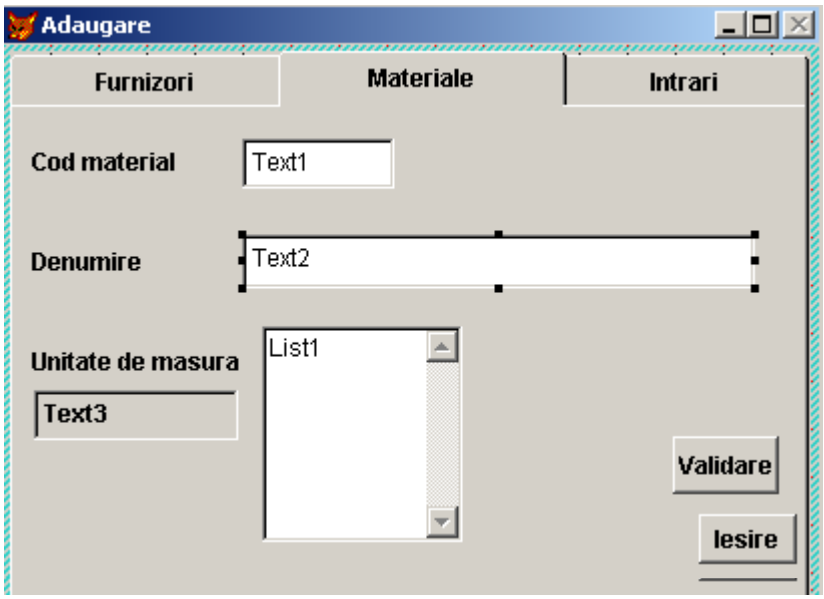


Fig. 6

Se stabilesc obiectele pentru pagina **Page2** (proprietatea **Caption**='*Materiale'*): **Text1, Text2, Text3, List1 (**listă), un buton de comandă (**Command1**) şi trei etichete **Label1, Label2, Label3**, (Figura 6). Ordinea obiectelor se stabileşte din meniul **View**, submeniul **TabOrder**.

În continuare se stabilesc câteva proprietăți referitoare la obiectele alese:

- pentru obiectul **Page2** (*Materiale*) se accesează codul procedurilor ataşate diferitelor evenimente utilizând butonul dreapta al mouse-ului şi alegând din meniu opțiunea **Code**. Se selectează evenimentul **Activate** şi se adăugă următoarele linii de cod în editorul deschis:

```
SELECT 2
CODM=0
DENM=' '
THISFORM.PAGEFRAME1.PAGE2.REFRESH
THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS
```

- pentru obiectul **Page2** (*Materiale*) se selectează procedura **Init** la care se ataşează următoarele linii de cod (se inițializează şi introduc valorile într-un vector care va fi utilizat ulterior pentru obiectul **list1** referitor la unitatea de măsură)

```
PUBLIC UMAS(6)
UMAS(1)='BUC'
UMAS(2)='KG'
UMAS(3)='TO'
UMAS(4)='MC'
UMAS(5)='MP'
UMAS(6)='ML'
```

- pentru obiectul **Page2** se selectează procedura **Activate** la care se ataşează următorul cod

```
SELECT 2
CODM=0
DENM=' '
THISFORM.PAGEFRAME1.PAGE2.REFRESH
THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS
```

- pentru obiectul **text1** se specifică în fereastra de proprietăți, secțiunea **Data**, proprietatea **ControlSource**, variabila *codm*.
- pentru obiectul **Text2** se specifică la proprietatea **ControlSource** variabila *denm*.
- pentru obiectul text3 se specifică în fereastra de proprietăți: **ReadOnly=.T.** şi **FontBold=.T.** care va fi utilizat pentru verificarea alegerii unității de măsură din obiectul *list1*.
- pentru obiectul **List1** se alege din fereastra de proprietăți **RowSource**, unde se declară variabila *umas,* iar la **RowSourceType** se alege **Array** şi se ataşă următotorul cod pe evenimentul **Click**:

```
WITH THISFORM.PAGEFRAME1.PAGE2
    TEXT3.VALUE=.LIST1.VALUE
    TEXT3.REFRESH
    COMMAND1.SETFOCUS
```

```
ENDWITH
```

prin care se preia valoarea selectată în **List1** şi se afişează în **Text3**. S-a folosit combinația **With...EndWith** pentru a adresa proprietățile obiectelor **List1, Text3** şi **Command1** din **Page2** într-o formă mai scurtă (calificare globală).

- pentru obiectul command1 din lista de proprietăți se stabilesc valorile pentru **Caption**=*validare* şi **Fontbold= True**. Din lista de evenimente se selectează **Click Event** pentru care se ataşează următoarele linii de cod:

```
LOCATE FOR CODM=COD_MAT
IF FOUND(2) THEN
    WAIT WINDOW 'COD MATERIAL DUPLICAT !'
ELSE
    LOCATE FOR UPPER(ALLTRIM(DENM))=UPPER(ALLTRIM(DENUMIRE))
    IF FOUND(2) THEN
        WAIT WINDOW 'DENUMIRE DUPLICAT !'
    ELSE
        APPEND BLANK
        REPLACE COD_MAT WITH CODM,DENUMIRE WITH DENM,;
        UM WITH THISFORM.PAGEFRAME1.PAGE2.LIST1.VALUE
    ENDIF
ENDIF
CODM=0
DENM=''
THISFORM.PAGEFRAME1.PAGE2.REFRESH
THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS
```

Secvența de program realizează adăugarea în fişierul de materiale a valorilor introduse, care se face în uma validării codului şi denumirii materialului. Se verifică existența duplicatelor, dacă acestea există vor fi semnalate prin mesaje specifice şi adăugarea nu se efectuează.

Tabela *Intrari*

Stabilirea obiectelor pentru ultima pagină **Page3** (proprietatea **Caption**='*Intrari'*, **FontBold= True**) din setul de pagini **PageFrame1**. Se vor insera următoarele obiecte: **List1, List2, Text1, Text2, Text3, Text4, Text5, Command1** şi şapte etichete, de la **Label1** la **Label7**, Figura 7.

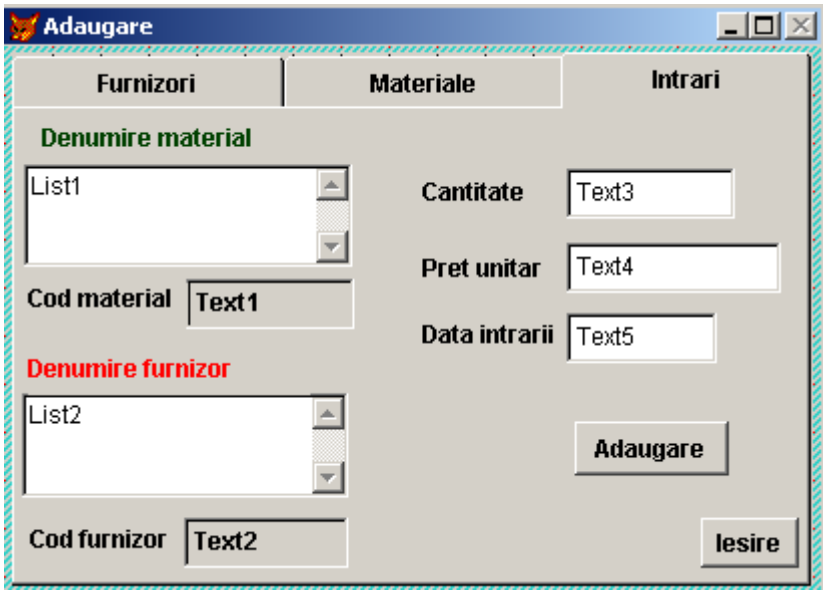


Fig. 7

Se stabilesc proprietăţile pentru obiectele declarate în videoformat:

- pentru obiectul **Page3** (*Intrari*) se accesează codul procedurilor ataşate diferitelor evenimente utilizând butonul dreapta al mouse-ului şi alegând din meniu opţiunea **Code**. Se selectează procedura **Activate** şi se adăugă următoarele linii de cod în editorul deschis

```
SELECT 2
GO TOP
SELECT 3
GO TOP
SELECT 4
PUN=0
CANTIT=0
CODM=MATERIALE.COD_MAT
CODF=FURNIZORI.COD_FURN
DATAIN={ / / }
THISFORM.PAGEFRAME1.PAGE3.LIST1.SETFOCUS
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

Procedura selectează tabelele care vor fi folosite, inițializează variabilele pentru cantitate, preț unitar, cod material, cod furnizor şi data intrării. Se trimite focusul pe obiectul L**ist1** şi se actualizează pagina cu noile valori.

- pentru obiectul **List1** se alege din fereastra de proprietăți **RowSource***,* unde se declară *materiale*, **RowsourceType**, se alege **fields, ControlSource**, se declară *denumire* (câmpul cu acelaşi nume ca în fişierul *materiale.dbf*). Pentru evenimentul **Click** se asociază următoarele linii de cod

```
SELECT 2
CODM=COD_MAT
THISFORM.PAGEFRAME1.PAGE3.TEXT1.REFRESH
```

- pentru obiectul **List2** se alege din fereastra de proprietăți **RowSource***,* aici se declară *furnizori,* **RowsourceType**, se alege **Fields, ControlSource** se declară *denumire* (câmpul cu acelaşi nume al fişierului *furnizori.dbf*). Evenimentul **Click** are ataşate următoarele linii de cod

```
SELECT 3
CODF=COD_FURN
THISFORM.PAGEFRAME1.PAGE3.TEXT2.REFRESH
THISFORM.PAGEFRAME1.PAGE3.TEXT3.SETFOCUS
```

- pentru obiectul *text1* se introduce în fereastra de proprietăți la **ControlSource** valoarea *codm* (variabila în care se preia din lista **List1** codul materialului), **FontBold= True** şi **ReadOnly True**.

- pentru obiectul **Text2** se introduce în fereastra de proprietăți la **ControlSource** valoarea *codf* (variabila în care se preia din lista **List2** codul furnizorului), **FontBold= True** şi **ReadOnly True.**

- pentru obiectul **Text3** se introduce în fereastra de proprietăți la **ControlSource** valoarea *cantit* (variabila în care se preia cantitatea);

- pentru obiectul *text4* se introduce în fereastra de proprietăți la **ControlSource** valoarea *pun* (variabila în care se preia prețul unitar);
- pentru obiectul **Text5** se introduce în fereastra de proprietăți la **ControlSource** valoarea *datain* (variabila în care se preia data intrării);
- pentru obiectul **Command1** la proprietatea **Caption** se introduce *validare*, **FontBold= True***,* iar la evenimentul **Click** se introduce secvența de cod

```
SELECT 4
APPEND BLANK
REPLACE COD_FURN WITH CODF, COD_MAT WITH CODM, CANT WITH CANTIT, PU
WITH PUN,;
DATA_IN WITH DATAIN
PUN=0
CANTIT=0
SELECT 2
GO TOP
CODM=MATERIALE.COD_MAT
SELECT 3
GO TOP
CODF=FURNIZORI.COD_FURN
DATAIN={  /  /  }
THISFORM.PAGEFRAME1.PAGE3.REFRESH
THISFORM.PAGEFRAME1.PAGE3.LIST1.SETFOCUS
```

Secvența de program realizează selectarea fişierului *intrari.dbf* şi adăugarea conținutului variabilelor ataşate obiectelor din videoformat la câmpurile fişierului. Se reiniționalizează variabilele, se actualizează câmpurile cu noile valori şi se trimite focus-ul pe obiectul **List1**.

Etichetele vor avea fixate valorile pentru următoarele proprietăți:

Obiect	Caption	FontBold	ForeColor
Label1	Cod material	true	0,0,0

Label2	Cod furnizor	true	0,0,0
Label3	Cantitate	true	0,0,0
Label4	Pret unitar	true	0,0,0
Label5	Data intrarii	true	0,0,0
Label6	Denumire material	true	0,64,0
Label7	Denumire furnizor	true	255,0,0

Videoformatul *Modificare* pentru tabela *Furnizori*

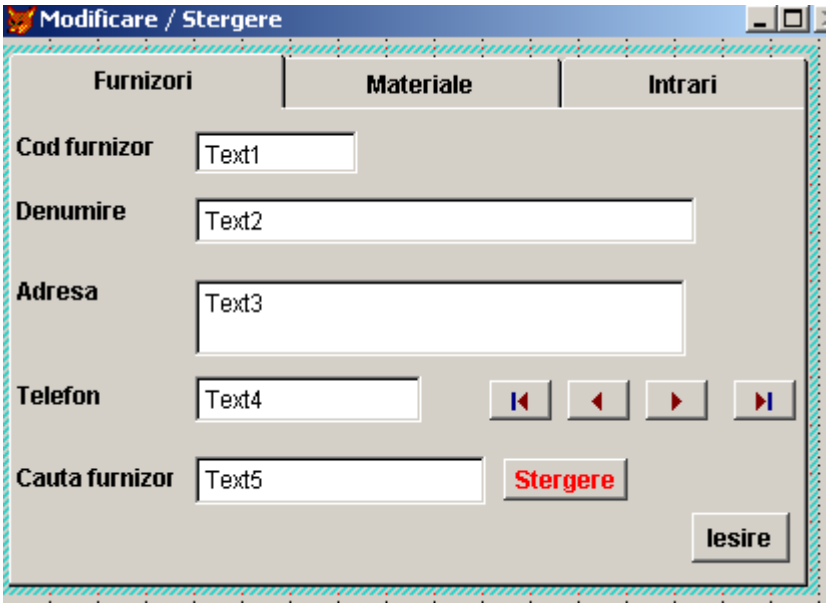


Fig. 8

Aceast videoformat are numele intern **Form1**, numele extern este *modificare.scx,* iar pentru proprietatea **Caption** se alege textul *Modificare*, Figura 8.

Butonul *Iesire* este primul strat şi se aplica deasupra obiectului **PageFrame1** (de tip container având proprietatea **PageCount** =3); pe butonul *Iesire* la evenimentul **Click** se asociază comanda THISFORM.RELEASE.

Se face click cu butonul din dreapta al mouse-ului şi se alege opţiunea *Edit* pentru a accesa elementele containerului, respectiv cele trei pagini: *Furnizori*, *Materiale*, *Intrari*. Numele extern şi grosimea de caracter a paginilor se stabileşte utilizând proprietăţile **Caption** şi **FontBold= True**.

Adresarea paginilor se realizează făcând click cu butonul stâng al mouse-ului pe fiecare pagină.

La fel ca şi la videoformatul precedent numele intern al paginii poate fi schimbat, dar vor fi păstrate cele implicite (**Page1, Page2, Page3**).

Pentru stabilirea obiectelor pentru prima pagină **Page1** (proprietatea **caption**=*'Furnizori'*) din setul de pagini **PageFrame1** se procedează astfel: se inserează următoarele obiecte **Text1, Text2, Text3, Text4, Ttext5,** cinci butoane de comandă (**Command1..Command5**) şi cinci etichete (**Label1..Label5**).

Stabilirea proprietăților pentru obiectele alese se realizează astfel:

- pentru obiectul **Page1** (*Furnizori*) se accesează codul procedurilor ataşate diferitelor evenimente utilizând butonul dreapta al mouse-ului şi alegând din meniu opțiunea **Code**. Se selectează evenimentul **Activate** şi se adăugă următoarele linii de cod în editorul deschis:

```
SELECT 3
GO TOP
DENF=' '
THISFORM.PAGEFRAME1.PAGE1.REFRESH
THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS
```

- pentru obiectele de tip text la proprietăți, secțiunea **Data**, se aleg pentru **ControlSource** numele câmpurilor din fişiserul *furnizori.dbf*: **Text1**- *cod_furn*, **Text2**- *denumire,* **Text3**- *adresa,* **Text4**- *tel,* iar pentru **Text5**- *denf,* care este variabila de preluare a şirului de caractere utilizat pentru căutarea în fişier a denumirii furnizorului.
- Obiectul **Text5** va avea ataşat pe procedura **Valid** următoarele linii de cod pentru căutarea şirului de caractere preluat în variabila *codf*

```
LOCATE FOR UPPER(LEFT(DENF, LEN(ALLTRIM(DENF))))=UPPER(LEFT(DENUMIRE,
LEN(ALLTRIM(DENF))))
IF FOUND(3) THEN
    THISFORM.PAGEFRAME1.PAGE1.REFRESH
ELSE
    MESSAGEBOX('NU EXISTA !',64)
ENDIF
```

- Etichetele vor avea trecute la proprietatea **Caption** denumirile explicite ale variabilelor utilizate: *Cod furnizor (***Label1***), Denumire (***Label2***), Adresa (***Label3***), Telefon (***Label4***)* şi *Cauta furnizor (***Label5***)*. Proprietatea **FontBold** se setează pe valoarea **True**.
- Butonul de ştergere (nume intern **Command1**, proprietatea **Caption**=*'Stergere'*, **FontBold=True**, **ForeColo***r=255,0,0*) va avea ataşate următoare linii de cod pentru evenimentul **Click**

```
* VARIABILA CODF CAPĂTA VALOAREA CURENTĂ A CODULUI FURNIZORULUI
CODF=COD_FURN
* STERGE LOGIC ÎNREGISTRAREA CURENTĂ
DELETE
* STERGE FIZIC ÎNREGISTRAREA CURENTĂ
PACK
* SELECȚIA FIŞIERULUI DIN ZONA 4(INTRARI.DBF)
SELECT 4
* CAUTĂ ÎNREGISTRĂRILE CU CODF=COD_FURN
LOCATE FOR CODF=COD_FURN
* ATÂTA VREME CÂT GASEŞTI CODF
DO WHILE FOUND(4)
* ŞTERGE LOGIC ÎNREGISTRAREA
    DELETE
    CONTINUE
* SFÂRŞIT CICLU
ENDDO
* ŞTERGE FIZIC ÎNREGISTRAREA
PACK
* SELECȚIE FIŞIER DIN ZONA 3 (FURNIZORI.DBF)
SELECT 3
* ACTUALIZARE CÂMPURI PAGINA 1
THISFORM.PAGEFRAME1.PAGE1.REFRESH
* FOCUSUL(CONTROLUL) ESTE TRIMIS LA TEXT1.
THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS
```

- urmează grupul de patru butoane (**Command2..Command5**), utilizate pentru navigarea în cadrul fişierului *furnizori.dbf,* cu următoarele valori pentru declararea numelui icon-lui care va fi ataşat (proprietatea **Picture**)

Obiect	Picture
Command2	c:\proiect_ex\top.bmp
Command3	c:\proiect_ex\prev.bmp
Command4	c:\proiect_ex\next.bmp
Command5	c:\proiect_ex\end.bmp

Fiecare buton are ataşată câte o procedură pe evenimentul **Click** care execută deplasarea.

- Butonul **Command2** – deplasare pe prima înregistrare

```
GO TOP
THISFORM.PAGEFRAME1.PAGE1.REFRESH
```

- Butonul **Command3** – deplasare o înregistrare înapoi

```
SKIP -1
IF BOF(3)
    GO TOP
    MESSAGEBOX('INCEPUT FISIER !',64)
ENDIF
THISFORM.PAGEFRAME1.PAGE1.REFRESH
```

- Butonul **Command4** – deplasare o înregistrare înainte

```
SKIP 1
IF EOF(3)
    GO BOTTOM
    MESSAGEBOX('SFÂRSIT FISIER !',64)
ENDIF
THISFORM.PAGEFRAME1.PAGE1.REFRESH
```

Butonul **Command5** – deplasare la ultima înregistrare

GO BOTTOM

THISFORM.PAGEFRAME1.PAGE1.REFRESH

Videoformatul *Modificare* pentru tabela *Materiale*

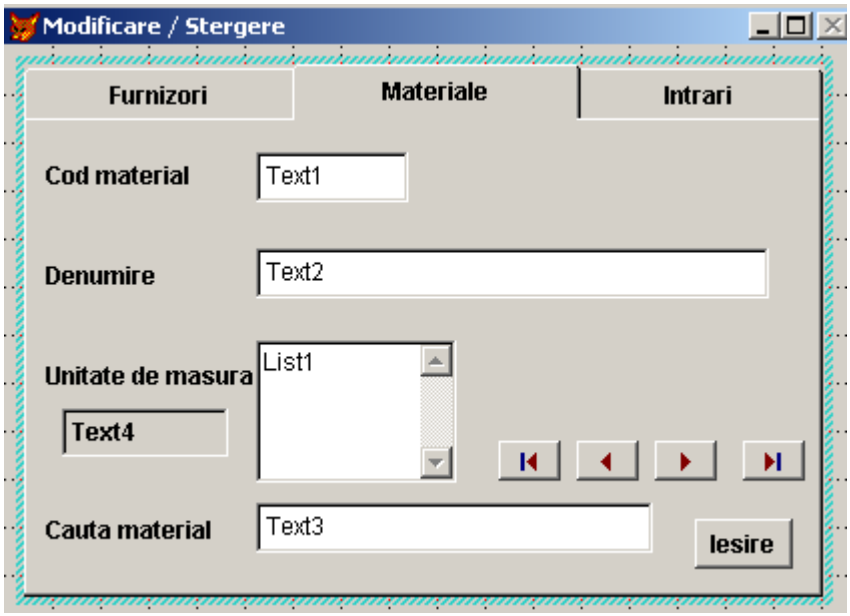


Fig. 9

Stabilirea obiectelor pentru pagina **Page2** (proprietatea **Caption**='*Materiale'*, **FontBold**= **True**): **Text1, Text2, Text3, Text4, List1** (listă), patru butoane de comandă (**Command2..Command5**) şi patru etichete **Label1, Label2, Label3, Label4**, Figura 9. Acțiunile ataşate se reazlizează după cum urmează.

- pentru obiectul **Page2** (*Materiale*) se accesează codul procedurilor ataşate diferitelor evenimente utilizând butonul dreapta al mouse-ului şi alegând din meniu opțiunea **Code**. Se selectează procedura **Init** şi se adăugă următoarele linii de cod în editorul deschis

```
PUBLIC UMAS(6)
UMAS(1)='BUC'
UMAS(2)='KG'
UMAS(3)='TO'
UMAS(4)='MC'
UMAS(5)='MP'
```

UMAS(6)='ML'

- pentru obiectul **Page2** se selectează procedura **Activate** la care se ataşează următorul cod

```
SELECT 2
GO TOP
DENM=' '
THISFORM.PAGEFRAME1.PAGE2.REFRESH
THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS
```

- pentru obiectele de tip text se stabilesc următoarele valori pentru prorietatățile **ControlSource** şi **ReadOnly**:

Obiect	ControlSource	ReadOnly
Text1	Cod_mat	.F.
Text2	Denumire	.F.
Text3	denm	.F.
Text4	Um	.T.

Proprietatea **ReadOnly=.F.** este implicită.

- Obiectul **Text3** va avea ataşat următorul cod la procedura **Valid**

```
LOCATE FOR UPPER(LEFT(DENM,LEN(ALLTRIM(DENM))))=;
UPPER(LEFT(DENUMIRE,LEN(ALLTRIM(DENM))))
IF FOUND(2) THEN
    THISFORM.PAGEFRAME1.PAGE2.LIST1.VALUE=UM
    THISFORM.PAGEFRAME1.PAGE2.REFRESH
ELSE
    MESSAGEBOX('NU EXISTA !',64)
ENDIF
```

- Obiectul **Text4** se utilizează pentru a afişa unitatea de măsură care este declarată în câmpul *um* al fişierului *materiale.dbf* şi pentru a afişa noua modificare în cazul operării în **List1**.

Secvența de cod realizezază căutarea materialului după şirul de caractere preluat în variabila *denm*, dacă găseşte o înregistrare cu valoarea câmpului *denumire=denm* se afişează datele, dacă nu, apare un mesaj specific.

- pentru obiectul **List1** vom alege din fereastra de proprietăți **RowSource** unde declarăm variabila *umas* iar la **RowsourceType** alegem A**array** şi ataşăm următorul cod pe evenimentul **Click:**

```
WITH THISFORM.PAGEFRAME1.PAGE2
    TEXT4.VALUE=.LIST1.VALUE
    UM=.TEXT4.VALUE
    TEXT4.REFRESH
    TEXT3.SETFOCUS
ENDWITH
```

Secvența realizează actualizarea valorii stocate în obiectului **Text4** cu valoarea selectată din **List1** în cazul unei modificări, noua valoare se scrie în câmpul *um* din fişierul *intrari.dbf* şi trimiterea controlului la următorul obiect **Text3** (căutare material).

- pentru cele patru etichete se stabilesc valorile pentru proprietățile **Caption** şi **FontBold**

Obiect	Caption	FontBold
Label1	Cod material	true
Label2	Denumire	true
Label3	Unitate de masura	true
Label4	Cauta material	true

- cele patru butoane de comandă (**Command2..Command5**) se utilizează pentru navigarea în fişierul *materiale.dbf* au următoarele valori pentru declararea numelui icon-lui care va fi ataşat:

Obiect	Picture
Command2	c:\proiect_ex\top.bmp
Command3	c:\proiect_ex\prev.bmp

Command4	c:\proiect_ex\next.bmp
Command5	c:\proiect_ex\end.bmp

Se ataşează următoarele linii de cod la evenimentul **Click**

Command2 – deplasare la începutul fişierului:

```
GO TOP
THISFORM.PAGEFRAME1.PAGE2.REFRESH
```

Command3 – deplasare o înregistrare înapoi:

```
SKIP -1
IF BOF(2)
    GO TOP
    MESSAGEBOX('INCEPUT FISIER !',64)
ENDIF
THISFORM.PAGEFRAME1.PAGE2.REFRESH
```

Command4 – deplasare o înregistrare înainte:

```
SKIP 1
IF EOF(2)
    GO BOTTOM
    MESSAGEBOX('SFÂRSIT FISIER !',64)
ENDIF
THISFORM.PAGEFRAME1.PAGE2.REFRESH
```

Command5 – deplasare la sfârşitul fişierului:

```
GO BOTTOM
THISFORM.PAGEFRAME1.PAGE2.REFRESH
```

Videoformatul *Modificare* pentru tabela *Intrari*

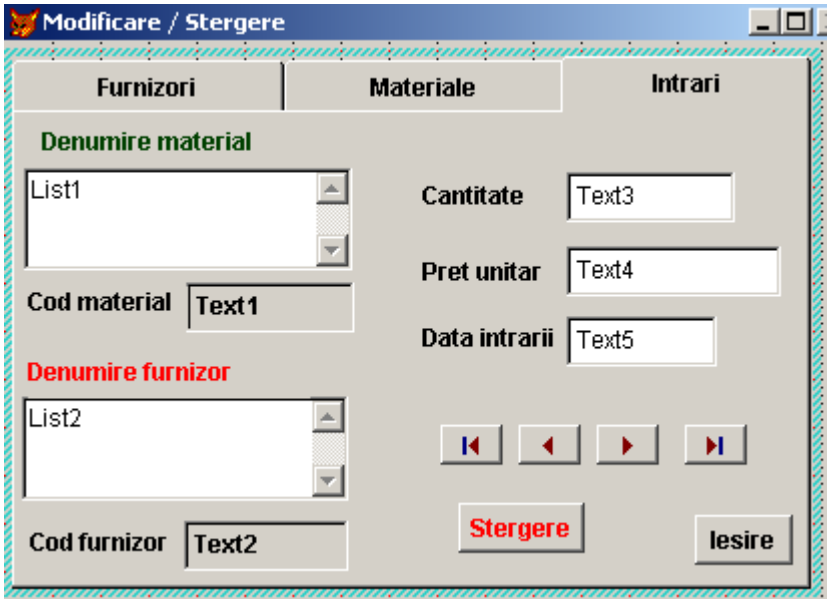


Fig. 10

În continuare se stabilesc obiectelor pentru ultima pagină **Page3** (proprietatea **Captio***n*=*'Intrari'*, **FontBold=True**) din setul de pagini **PageFrame1**. Următoarele obiecte vor fi inserate în acest videoformat : **List1, List2, Text1, Text2, Text3, Text4, Text5**, cinci butoane de comandă (**Command1..Command5**) şi şapte etichete **Label1..Label7**, Figura 10.

Pentru stabilirea proprietăților pentru obiectele declarate în videoformat se procedează astfel:

- pentru obiectul **Page3** (*Intrari*) se accesează codul procedurilor ataşate diferitelor evenimente Se selectează procedura **Activate** şi se adăugă următoarele linii de cod în editorul deschis

```
SELECT 2
GO TOP
SELECT 3
GO TOP
SELECT 4
GO TOP
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

- pentru obiectul **List1** - se alege din fereastra de proprietăți **RowSource**, unde se declară *materiale*, **RowsourceType**, se alege **Fields**, **ControlSource**, se declară *denumire* (câmpul cu acelaşi nume ca în fişierului *materiale.dbf*) şi se ataşeză la evenimentul **Click** codul:

```
SELECT 2
CODM=COD_MAT
THISFORM.PAGEFRAME1.PAGE3.TEXT1.REFRESH
```

- pentru obiectul **List2** se alege din fereastra de proprietăţi **RowSource**, unde se declară *furnizori,* **RowsourceType**, se alege F**ields**, **ControlSource**, se declară *denumire* (câmpul cu acelaşi nume ca în fişierul *furnizori.dbf*) şi se ataşează la evenimentul **Click** codul:

```
SELECT 3
CODF=COD_FURN
THISFORM.PAGEFRAME1.PAGE3.TEXT2.REFRESH
THISFORM.PAGEFRAME1.PAGE3.TEXT3.SETFOCUS
```

- pentru obiectul **Text1** se introduce în fereastra de proprietăţi la **ControlSource** valoarea *cod_mat* (câmpul din fişier corespunzător codului de material), **FontBold= True** şi **ReadOnly=True**.
- Pentru obiectul **Text2** se introduce în fereastra de proprietăţi la **ControlSource** valoarea *cod_furn* (câmpul din fişier corespunzător codului de furnizor), **FontBold=True** şi **ReadOnly= True.**
- pentru obiectul **Text3** se introduce în fereastra de proprietăţi la **ControlSource** valoarea *cant* (câmpul în care este stocată cantitatea);
- pentruobiectul **Text4** se introduce în fereastra de proprietăţi la **ControlSource** valoarea *pu* (câmpul în care este stocat preţul unitar);
- pentru obiectul **Text5** se introduce în fereastra de proprietăţi la **ControlSource** valoarea *data_in* (câmpul în care este stocată data intrării);
- pentru butonul **Command1** se stabilesc următoarele proprietăţi: **Caption**=*'Stergere',* **FontBold= True**, **Forecolor**=*255,0,0* şi se ataşează următorul cod la evenimentul **Click**:

```
BTNVALUE=0
BTNVALUE=MESSAGEBOX('STERGERE ?',4+32+256)
IF BTNVALUE=6 THEN
    DELETE
    PACK
```

```
ENDIF
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

Secvența de cod realizează ştergerea înregistrării curente în urma unei confirmări suplimentare solicitate prin cutia de dialog **Messagebox**; valoarea returnată de cutia de dialog preluată în variabila *btnvalue* este testată şi dacă s-a ales butonul **OK** se realizează ştergerea. Parametrii cutiei de dialog **Messagebox** sunt:

MESSAGEBOX(*cMessageText* [, *nDialogBoxType* [, *cTitleBarText*]])

cMessageText -textul care va fi afişat;

nDialogBoxType – specifică tipurile de butoane ,butonul implicit şi icon-ul afişat în antet;

cTitleBarText – titlul cutiei de dialog (antet) .

Valorile pentru *nDialogBoxType* sunt date în tabelul de mai jos:

Valoare	Tipuri butoane
0	Numai butonul OK
1	OK şi Cancel
2	Abort, Retry, and Ignore
3	Yes, No şi Cancel
4	Yes şi No
5	Retry şi Cancel

Valoare	Icon
16	Stop
32	Semnul întrebării
48	Semnul exclamării
64	Semnul Informație (I)

Valoare	Buton implicit
0	Primul buton
256	Al doilea buton
512	Al treilea buton

(4- butoanele **YES** şi **NO,** 32- tipul de icon afişat- semnul întrebării şi 256- butonul doi, **NO** este implicit).

- cele patru butoane (**Command2..Command5**) se utilizează pentru parcurgerea înregistrărilor din fişierul *intrari.dbf* având proprietatea **Picture** cu următoarele valori:

Obiect	Picture
Command2	c:\proiect_ex\top.bmp
Command3	c:\proiect_ex\prev.bmp
Command4	c:\proiect_ex\next.bmp
Command5	c:\proiect_ex\end.bmp

la care se ataşează următorul cod:

Command2 -deplasare la începutul fişierului:

```
SELECT 4
GO TOP
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

Command3 – deplasare o înregistrare înapoi:

```
SELECT 4
SKIP -1
IF BOF(4)
    GO TOP
    MESSAGEBOX('INCEPUT FISIER !',64)
ENDIF
SELECT 4
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

Command4 – deplasare o înregistrare înainte:

```
SELECT 4
SKIP 1
```

```
IF EOF(4)
    GO BOTTOM
    MESSAGEBOX('SFÂRSIT FISIER !',64)
ENDIF
SELECT 4
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

Command5 – deplasare la sfârşitul fişierului:

```
SELECT 4
GO BOTTOM
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

- cele şapte etichete au stabilite următoarele proprietăţi:

Obiect	Caption	FontBold	ForeColor
Label1	Cod material	true	0,0,0
Label2	Cod furnizor	true	0,0,0
Label3	Cantitate	true	0,0,0
Label4	Pret unitar	true	0,0,0
Label5	Data intrarii	true	0,0,0
Label6	Denumire material	true	0,64,0
Lab6el7	Denumire furnizor	true	255,0,0

Videoformatul *mat_data.scx* pentru lista materiale/Data

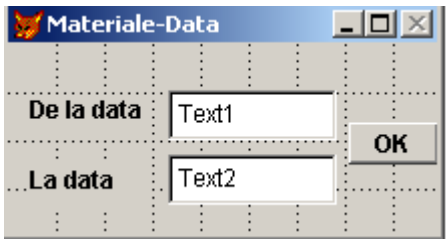


Fig. 11

Acest videoformat are 5 obiecte: două obiecte **TextBox** *(***Text1, Text2***)*, două etichete (**Label1, Label2**) şi un buton de comandă (**Command1**) , Figura 11.

Proprietățile pentru aceste obiecte sunt fixate astfel:

- procedura ataşată evenimentului **Init** este:

```
PUBLIC DATAIN1,DATAIN2
DATAIN1={ / / }
DATAIN2={ / / }
```

Se utilizează două variabile pentru a prelua intervalul calendaristic pentru care se face selecția înregistrărilor din fişierul *intrari.dbf*.

- procedura ataşată evenimentului **Activate** este:

```
DATAIN1={ / / }
DATAIN2={ / / }
THISFORM.TEXT1.SETFOCUS
THISFORM.REFRESH
```

Se inițializează cele două variabile declarate anterior şi se trimite controlul catre obiectul **Text1**.

- cele două etichete **Label1** şi **Label2** vor avea stabilite următoarele proprietatăți:

Obiect	Caption	FontBold	ForeColor
Label1	De la data	true	0,0,0
Label1	La data	true	0,0,0

Text1 şi **Text2** vor avea declarat pentru **ControlSource** *datain1* şi *datain2*.
Command1 va avea ataşat pe evenimentul **Click** următoarea secvență de cod

```
IF DATAIN1>DATAIN2 THEN
```

```
    MESSAGEBOX('A DOUA DATÃ CALENDARISTICÃ TREBUIE SÃ FIE MAI MARE
DECÂT PRIMA !',48)
    DATAIN2={  /  /  }
    THISFORM.TEXT2.SETFOCUS
    THISFORM.REFRESH
ELSE
    SELECT * FROM INTRARI INTO TABLE TEMP1 HAVING
BETWEEN(DATA_IN,DATAIN1,DATAIN2) ORDER BY DATA_IN ;    ASC
    ALTER TABLE TEMP1 DROP COLUMN COD_FURN
    ALTER TABLE TEMP1 ADD COLUMN DENUMIRE C(40)
    USE
    SELECT 5
    USE TEMP1
    FOR I=1 TO RECCOUNT(5)
        GOTO I
        CODM=COD_MAT
        SELECT 2
        LOCATE FOR CODM=COD_MAT
        IF FOUND(2) THEN
            DENM=DENUMIRE
        ENDIF
        SELECT 5
        REPLACE DENUMIRE WITH DENM
    ENDFOR
    SELECT 5
    REPORT FORM MATER_INTR PREVIEW
    USE
    DELETE FILE 'TEMP1.DBF'
    THISFORM.RELEASE
ENDIF
```

Secvenţa de cod realizează validarea intervalului de timp comparând *datain1* cu *datain2*. în cazul în care *datain1* > *datain2* intervalul nu este valid, se afişează un mesaj explicativ, se reiniţializează variabila *datain2* şi controlul este trimis la obiectul **Text2**.

În cazul în care perioada este validată se realizeză o selecție din fişierul *intrari.dbf* în fişierul *temp1.dbf* a tuturor înregistrărilor care respectă condiția ca data calendaristică stocată în câmpul *data_in* să fie cuprinsă în perioada care este preluată în variabilele *datain1* şi *datain2*, ordonate ascendent după câmpul *data_in*.

Se şterge coloana *cod_furn* din fişierul *temp1.dbf* şi se adaugă o nouă coloană cu numele *denumire* de tip caracter (se utilizează comanda SQL ALTER TABLE).

Se selectează zona de lucru 5 şi se deschide tabela *temp.dbf* şi într-un ciclu FOR se parcurg secvențial înregistrările din acest fişier concomitent cu căutarea denumirii materialului în fişierul *materiale.dbf* deschis în zona de lucru 2 (pe baza câmpului comun *cod_mat*). Se transferă conținutul câmpului *denumire* din fişierul *materiale.dbf* în câmpul *denumire* din fişierul *temp1.dbf*.

La ieşirea din ciclu se lansează execuția raportului *mater_intr.frx* cu datele fişierului *temp1.dbf*. După terminarea execuției raportului, se şterge fişierul *temp1.dbf* şi se dă controlul meniului *menu1.mpr*.

Raportul în **Design View** are următoarea structură, Figura 12.

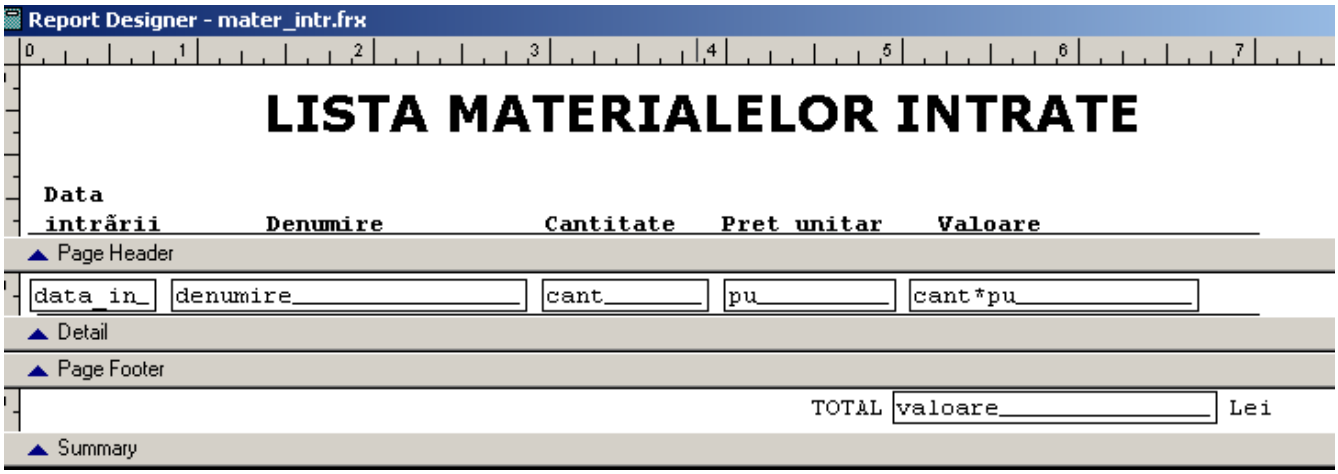


Fig. 12

În meniul **Report** se alege opțiunea **Title/Summary** şi în fereastra apărută, Figura 13, se selectează **Summary band** care va asigura tipărirea câmpurilor imediat ce se termină raportul; în caz contrar afişarea se va face în josul paginii (**Page Footer**).

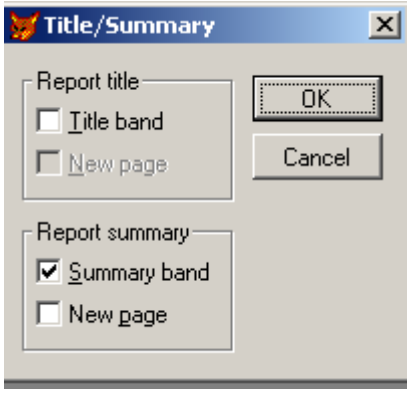


Fig. 13

În meniul **Report** la opțiunea **Variables**, se defineşte variabila *valoare* în care se va stoca suma produselor *pu*cant,* obținute de la fiecare înregistrare pentru a afişa o valoare totală a materialelor achiziționate într-o anumtă perioadă de timp, Figura 14.

În secțiunea **Page Header** sunt şase etichete corespunzătoare titlului raportului şi capului de tabel pentru care se afişează datele din fişierul *temp1.dbf*.

În secțiunea **Detail** sunt patru câmpuri de editare corespunzătoare datelor din fişier (*data_in, denumire, cant, pu*) şi un câmp utilizat pentru a afişa valoarea mărfii (*pu*cant*) pentru fiecare înregistrare.

În secțiunea **Page Footer** nu se trece nimic.

În secțiunea **Summary** se declară câmpul pentru afişarea variabilei *valoare* care sumează produsele parțiale *pu*cant*.

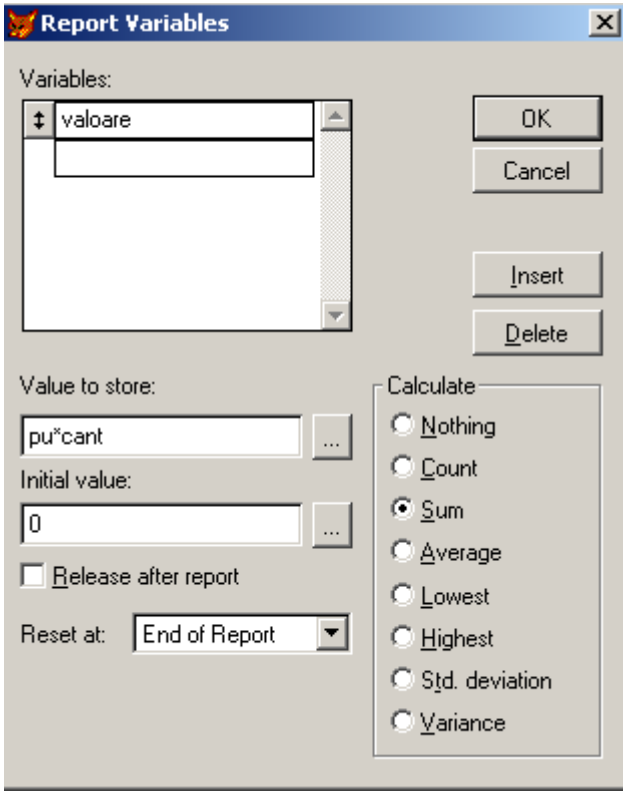


Fig. 14

Opţiunea *Lista Furnizori/Materiale* apelează în secvenţa de cod ataşată raportul *furniz_mater.frx* care are următoarea structură în modul **Design View**, Figura 15:

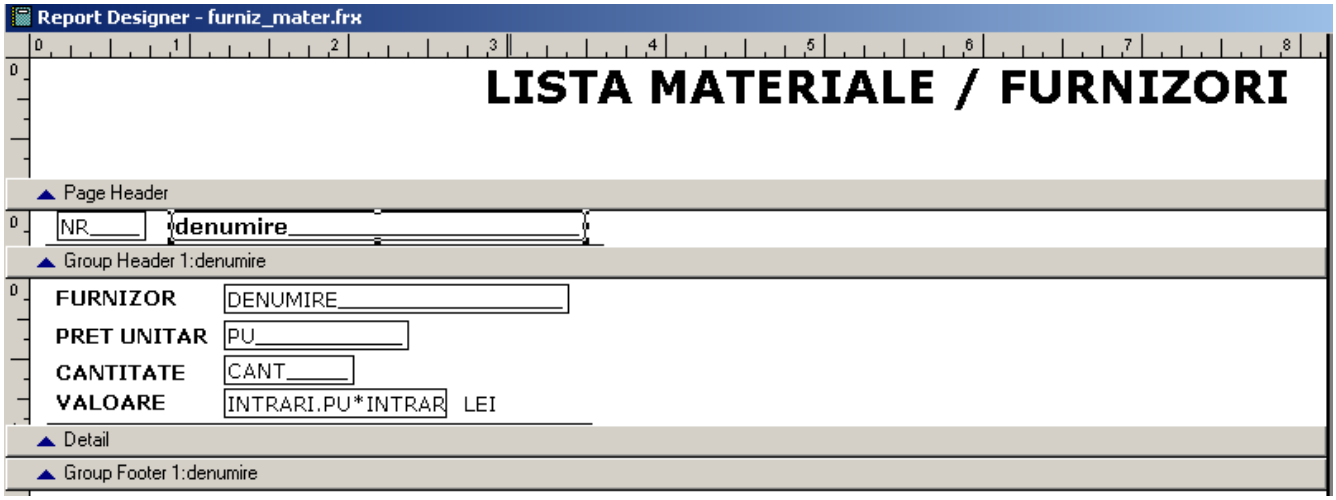


Fig. 15

În meniul **Report**, opţiunea Data Grouping, se declară câmpul *denumire* din fişierul *materiale.dbf,* după care se va face gruparea datelor, utilizându-se adresarea prin calificare

(*nume_fişier.câmp*, respectiv *materiale.denumire*) Figura 16. Opțiunea se foloseşte pentru a parcurge secvențial înregistrările din fişier la care se vor asocia şi tipări, pe baza relațiilor definite în celelalte două fişiere, datele corespunzătoare.

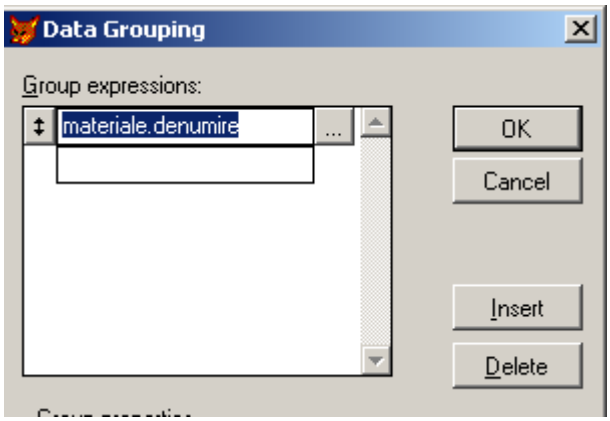


Fig. 16

Din meniul **Report** se selectează opțiunea **Variables** şi se defineşte variabila *NR,* iar la opțiunea **Calculations**, se selectează **Count**, Figura 17. Variabila se utilizează pentru numărarea înregistrărilor tipărite obținându-se astfel numărul curent.

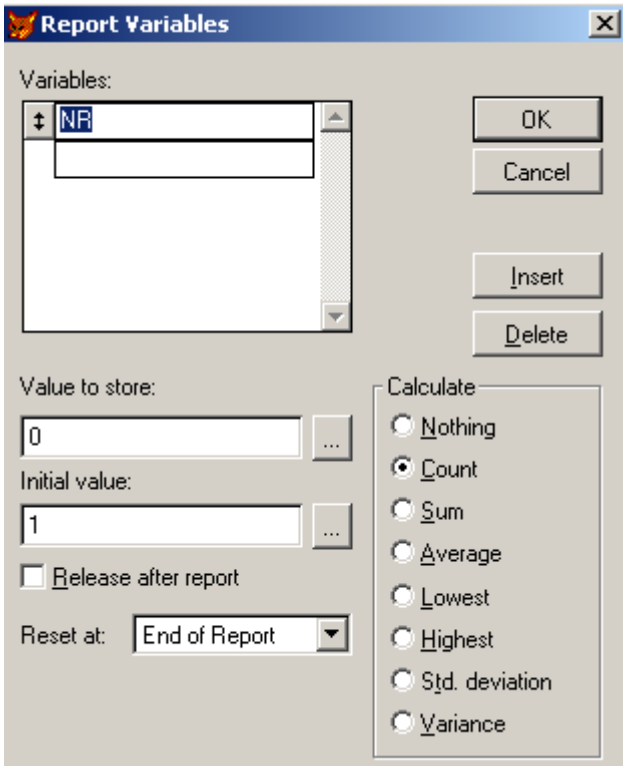


Fig. 17

Ca efect al opțiunii selectate (**Data Grouping**), raportul se împarte în mai multe secțiuni, astfel:

- secțiunea **Header** în care este declarată sub formă de etichetă denumirea raportului (se alege din meniul **Format** opțiunea **Font** şi se stabileşte **Verdana**, iar pentru **FontStyle=Bold, Size**=20);
- secțiunea **Group Header1***: denumire* în care se declară variabila *NR* şi câmpul de grupare, respectiv *materiale.denumire*. Pentru aceste câmpuri din meniul **Format**, la opțiunea **Font**, se alege **Verdana** şi **FontStyle=Regular** şi **Bold** (pentru *materiale.denumire*). Se observă că există posibilitatea de a definii mai multe câmpuri de grupare;
- secțiunea **Detail** se prevăd patru etichete pentru (*furnizor, pret unitar, cantitate, valoare*) şi patru câmpuri pentru afişarea datelor din fişierele între care s-au stabilit relații: *furnizori.denumire*, *intrari.pu, intrari.cant* şi produsul *intrari.pu*intrari.pu*. Pentru aceste câmpuri caracterul de tipărire este **Verdana** şi stilul caracterului **Regular**;
- în secțiunea **Group Footer1***: denumire* nu se trece nimic.

În continuare este prezentat codul sursă generat de wizard pentru aplicația de mai sus.

```
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\INIT_VAR.PRG
*:*****************************************************************************
*:
*: Procedure File C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\DOC\INIT_VAR.PRG
*:
*:
*:
*:
*:
*:
*:
*:
*:
*:
*:
*:
*:
*:
*:
```

```
*: Documented using Visual FoxPro Formatting wizard version  .05
*:*****************************************************************************
*:   INIT_VAR
PUBLIC codf,denf,adr,tl,codm,denm,pun,datain,cantit
SET date to british
codf=0
denf=''
adr=' '
tl=''
codm=0
denm=''
pun=0
datain={  /  /  }
cantit=0
CLOSE databases
IF empty(sys(2000,'materiale.dbf'))
    CREATE table materiale (cod_mat n(5),denumire c(40),um c(6))
ELSE
    USE materiale
ENDIF
INDEX on cod_mat tag cod_mat additive

IF empty(sys(2000,'intrari.dbf'))
    CREATE table intrari (cod_mat n(5),cod_furn n(4),cant n(6), pu
n(8,2),data_in d)
ELSE
    USE intrari
ENDIF

IF empty(sys(2000,'furnizori.dbf'))
    CREATE table furnizori (cod_furn n(4),denumire c(40),adresa m,tel c(10))
ELSE
    USE furnizori
ENDIF
INDEX on cod_furn tag cod_furn additive

CLOSE database
SELECT 2
USE materiale
SELECT 3
USE furnizori
```

```
SELECT 4
USE intrari
DO MENU1.MPR
READ EVENTS
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\ADAUG.SCX
Name = "Dataenvironment"
Top = -1
Left = -1
Height = 272
Width = 408
DoCreate = .T.
Caption = "Adaugare"
Name = "Form1"
ErasePage = .T.
PageCount = 3
Top = 3
Left = 4
Width = 400
Height = 266
TabIndex = 1
Name = "Pageframe1"
Page1.FontBold = .T.
Page1.Caption = "Furnizori"
Page1.Name = "Page1"
Page2.FontBold = .T.
Page2.Caption = "Materiale"
Page2.Name = "Page2"
Page3.FontBold = .T.
Page3.Caption = "Intrari"
Page3.Name = "Page3"
PROCEDURE Page1.Activate
    SELECT 3
    codf=0
    den=''
    adr=' '
    tl=''
    thisform.Pageframe1.Page1.refresh
    thisform.Pageframe1.Page1.text1.setfocus
ENDPROC
PROCEDURE Page2.Activate
```

```
    SELECT 2
    codm=0
    denm=''
    thisform.Pageframe1.Page2.refresh
    thisform.Pageframe1.Page2.text1.setfocus
ENDPROC
PROCEDURE Page2.Init
    PUBLIC umas(6)
    umas(1)='BUC'
    umas(2)='KG'
    umas(3)='TO'
    umas(4)='MC'
    umas(5)='MP'
    umas(6)='ML'


ENDPROC
PROCEDURE Page3.Activate
    SELECT 2
    GO top
    SELECT 3
    GO top
    SELECT 4
    pun=0
    cantit=0
    codm=materiale.cod_mat
    codf=furnizori.cod_furn
    datain={  /  /  }
    thisform.Pageframe1.Page3.list1.setfocus
    thisform.Pageframe1.Page3.refresh
ENDPROC
ControlSource = "codf"
Height = 21
Left = 104
TabIndex = 1
Top = 22
Width = 81
Name = "Text1"
ControlSource = "den"
Height = 23
Left = 103
```

```
TabIndex = 2
Top = 55
Width = 250
Name = "Text2"
ControlSource = "adr"
Height = 38
Left = 106
TabIndex = 3
Top = 96
Width = 245
Name = "Text3"
ControlSource = "tl"
Height = 23
Left = 106
TabIndex = 4
Top = 144
Width = 113
Name = "Text4"
Top = 157
Left = 337
Height = 26
Width = 53
FontBold = .T.
Caption = "Validare"
TabIndex = 5
Name = "Command1"
PROCEDURE Click
    LOCATE for codf=cod_furn
    IF found(3) then
        WAIT window 'Cod furnizor duplicat !'
    ELSE
        LOCATE for upper(alltrim(den))=upper(alltrim(denumire))
        IF found(3) then
            WAIT window 'Denumire duplicat !'
        ELSE
            APPEND blank
            REPLACE cod_furn with codf,denumire with den,adresa with adr,tel
with tl
        ENDIF
    ENDIF
    codf=0
```

```
    den=''
    adr=' '
    tl=''
    thisform.Pageframe1.Page1.refresh
    thisform.Pageframe1.Page1.text1.setfocus
ENDPROC
FontBold = .T.
Caption = "Cod furnizor"
Height = 21
Left = 19
Top = 22
Width = 72
TabIndex = 6
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 18
Top = 54
Width = 80
TabIndex = 7
Name = "Label2"
FontBold = .T.
Caption = "Adresa"
Height = 22
Left = 15
Top = 94
Width = 85
TabIndex = 8
Name = "Label3"
FontBold = .T.
Caption = "Telefon"
Height = 20
Left = 14
Top = 146
Width = 74
TabIndex = 9
Name = "Label4"
ControlSource = "codm"
Height = 25
Left = 115
```

```
TabIndex = 1
Top = 16
Width = 76
Name = "Text1"
ControlSource = "denm"
Height = 27
Left = 115
TabIndex = 2
Top = 64
Width = 256
Name = "Text2"
PROCEDURE Valid
    thisform.Pageframe1.Page2.list1.setfocus
ENDPROC
Top = 165
Left = 330
Height = 29
Width = 53
FontBold = .T.
Caption = "Validare"
TabIndex = 4
Name = "Command1"
PROCEDURE Click
    LOCATE for codm=cod_mat
    IF found(2) then
        WAIT window 'Cod material duplicat !'
    ELSE
        LOCATE for upper(alltrim(denm))=upper(alltrim(denumire))
        IF found(2) then
            WAIT window 'Denumire duplicat !'
        ELSE
            APPEND blank
            REPLACE cod_mat with codm,denumire with denm,;
                um with thisform.Pageframe1.Page2.list1.value
        ENDIF
    ENDIF
    codm=0
    denm=''
    thisform.Pageframe1.Page2.refresh
    thisform.Pageframe1.Page2.text1.setfocus
ENDPROC
```

```
FontBold = .T.
Caption = "Cod material"
Height = 24
Left = 10
Top = 20
Width = 82
TabIndex = 5
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 10
Top = 70
Width = 81
TabIndex = 6
Name = "Label2"
RowSourceType = 5
RowSource = "umas"
Height = 108
Left = 125
TabIndex = 3
Top = 110
Width = 100
Name = "List1"
PROCEDURE Click
    WITH thisform.Pageframe1.Page2
        .Text3.value=.list1.value
        .Text3.refresh
        .Command1.setfocus
    ENDWITH

ENDPROC
FontBold = .T.
Caption = "Unitate de masura"
Height = 22
Left = 10
Top = 120
Width = 105
Name = "Label3"
FontBold = .T.
Height = 25
```

```
Left = 11
ReadOnly = .T.
Top = 142
Width = 102
Name = "Text3"
FontBold = .T.
ControlSource = "codm"
Height = 25
Left = 86
ReadOnly = .T.
TabIndex = 1
Top = 85
Width = 84
Name = "Text1"
FontBold = .T.
ControlSource = "codf"
Height = 25
Left = 85
ReadOnly = .T.
TabIndex = 3
Top = 204
Width = 82
Name = "Text2"
ControlSource = "cantit"
Height = 25
Left = 276
TabIndex = 5
Top = 30
Width = 84
Name = "Text3"
ControlSource = "pun"
Height = 25
Left = 276
TabIndex = 6
Top = 67
Width = 107
Name = "Text4"
ControlSource = "datain"
Height = 25
Left = 276
TabIndex = 7
```

```
Top = 102
Width = 75
Name = "Text5"
BoundColumn = 1
RowSourceType = 6
RowSource = "materiale"
ControlSource = "denumire"
Height = 51
Left = 5
Sorted = .F.
TabIndex = 2
Top = 28
Width = 164
BoundTo = .T.
Name = "List1"
PROCEDURE Click
    SELECT 2
    codm=cod_mat
    thisform.Pageframe1.Page3.text1.refresh
ENDPROC
RowSourceType = 6
RowSource = "furnizori"
ControlSource = "denumire"
Height = 52
Left = 4
Sorted = .F.
TabIndex = 4
Top = 142
Width = 163
Name = "List2"
PROCEDURE Click
    SELECT 3
    codf=cod_furn
    thisform.Pageframe1.Page3.Text2.refresh
    thisform.Pageframe1.Page3.Text3.setfocus
ENDPROC
Top = 156
Left = 280
Height = 27
Width = 77
FontBold = .T.
```

```
Caption = "Adaugare"
TabIndex = 8
Name = "Command1"
PROCEDURE Click
    SELECT 4
    APPEND blank
    REPLACE cod_furn with codf, cod_mat with codm, cant with cantit,pu with
pun,;
        data_in with datain
    pun=0
    cantit=0
    SELECT 2
    GO top
    codm=materiale.cod_mat
    SELECT 3
    GO top
    codf=furnizori.cod_furn
    datain={  /  /  }
    thisform.Pageframe1.Page3.refresh
    thisform.Pageframe1.Page3.list1.setfocus
ENDPROC
FontBold = .T.
Caption = "Cod material"
Height = 23
Left = 7
Top = 87
Width = 81
TabIndex = 9
Name = "Label1"
FontBold = .T.
Caption = "Cod furnizor"
Height = 24
Left = 8
Top = 207
Width = 69
TabIndex = 10
Name = "Label2"
FontBold = .T.
Caption = "Cantitate"
Height = 28
Left = 204
```

```
Top = 34
Width = 57
TabIndex = 11
Name = "Label3"
FontBold = .T.
Caption = "Pret unitar"
Height = 28
Left = 204
Top = 72
Width = 66
TabIndex = 12
Name = "Label4"
FontBold = .T.
Caption = "Data intrarii"
Height = 26
Left = 204
Top = 105
Width = 69
TabIndex = 13
Name = "Label5"
FontBold = .T.
Caption = "Denumire material"
Height = 16
Left = 14
Top = 7
Width = 125
TabIndex = 14
ForeColor = 0,64,0
Name = "Label6"
FontBold = .T.
Caption = "Denumire furnizor"
Height = 22
Left = 7
Top = 122
Width = 142
TabIndex = 15
ForeColor = 255,0,0
Name = "Label7"
Top = 235
Left = 348
Height = 25
```

```
Width = 49
FontBold = .T.
Caption = "Iesire"
TabIndex = 2
Name = "Command1"
PROCEDURE Click
    thisform.release
ENDPROC
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\END.BMP
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\FURNIZ_MATER.FRX
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MAT_DATA.SCX
Name = "Dataenvironment"
Top = 1
Left = 67
Height = 94
Width = 214
DoCreate = .T.
Caption = "Materiale-Data"
Name = "Form1"
PROCEDURE Activate
    datain1={  /  /  }
    datain2={  /  /  }
    thisform.text1.setfocus
    thisform.refresh
ENDPROC
PROCEDURE Init
    PUBLIC datain1,datain2
    datain1={  /  /  }
    datain2={  /  /  }
ENDPROC
Top = 39
Left = 170
Height = 21
Width = 45
FontBold = .T.
Caption = "OK"
TabIndex = 3
Name = "Command1"
```

```
PROCEDURE Click
    IF datain1>datain2 then
        MESSAGEBOX('A doua data calendaristica trebuie sa fie mai mare decât
prima !',48)
        datain2={  /  /  }
        thisform.Text2.setfocus
        thisform.refresh
    ELSE
        SELECT * from intrari into table temp1 having
between(data_in,datain1,datain2) order by data_in asc
        ALTER table temp1 drop column cod_furn
        ALTER table temp1 add column denumire c(40)
        USE
        SELECT 5
        USE temp1
        FOR i=1 to reccount(5)
            GOTO i
            codm=cod_mat
            SELECT 2
            LOCATE for codm=cod_mat
            IF found(2) then
                denm=denumire
            ENDIF
            SELECT 5
            REPLACE denumire with denm
        ENDFOR

        SELECT 5
        REPORT form mater_intr preview
        USE
        Delete file 'temp1.dbf'
        thisform.release
    ENDIF
ENDPROC
ControlSource = "datain1"
Height = 25
Left = 80
TabIndex = 1
Top = 23
Width = 84
Name = "Text1"
```

```
ControlSource = "datain2"
Height = 25
Left = 80
TabIndex = 2
Top = 55
Width = 84
Name = "Text2"
FontBold = .T.
Caption = "De la data"
Height = 22
Left = 11
Top = 25
Width = 61
TabIndex = 4
Name = "Label1"
FontBold = .T.
Caption = "La data"
Height = 22
Left = 11
Top = 60
Width = 61
TabIndex = 5
Name = "Label2"
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MATER_INTR.FRX
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MENU1.MNX
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MENU1.MPR
*:*****************************************************************************
*:
*: Procedure File C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\DOC\MENU1.MPR
*:
*:
*:
*:
*:
*:
*:
*:
```

```
*:
*:
*:
*:
*:
*:
*:
*: Documented using Visual FoxPro Formatting wizard version  .05
*:*****************************************************************************
*:   MENU1
*:   _1j10vft5m
*:   _1j10vft5n
*:   _1j10vft5o
*:   _1j10vft5p
*:   _1j10vft5q
*:   _1j10vft5r
*:   _1j10vft5s
*       ***********************************************************
*       *
*       * 26/05/05              MENU1.MPR               14:40:05
*       *
*       ***********************************************************
*       *
*       * Author's Name
*       *
*       * Copyright (C) 2005 Company Name
*       * Address
*       * City,     Zip
*       *
*       * Description:
*       * This PROGRAM was automatically generated BY GENMENU.
*       *
*       ***********************************************************


*       ***********************************************************
*       *
*       *                       Menu Definition
*       *
*       ***********************************************************
```

```
SET SYSMENU TO
SET SYSMENU AUTOMATIC

DEFINE PAD _1j10vft5i OF _MSYSMENU PROMPT "Vizualizare BD" COLOR SCHEME 3 ;
    KEY ALT+V, ""
DEFINE PAD _1j10vft5j OF _MSYSMENU PROMPT "Actualizare BD" COLOR SCHEME 3 ;
    KEY ALT+A, ""
DEFINE PAD _1j10vft5k OF _MSYSMENU PROMPT "Rapoarte" COLOR SCHEME 3 ;
    KEY ALT+R, ""
DEFINE PAD _1j10vft5l OF _MSYSMENU PROMPT "Iesire" COLOR SCHEME 3 ;
    KEY ALT+i, ""
ON PAD _1j10vft5i OF _MSYSMENU ACTIVATE POPUP vizualizar
ON PAD _1j10vft5j OF _MSYSMENU ACTIVATE POPUP actualizar
ON PAD _1j10vft5k OF _MSYSMENU ACTIVATE POPUP rapoarte
ON SELECTION PAD _1j10vft5l OF _MSYSMENU ;
    DO _1j10vft5m ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")

DEFINE POPUP vizualizar MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF vizualizar PROMPT "Materiale"
DEFINE BAR 2 OF vizualizar PROMPT "Furnizori"
DEFINE BAR 3 OF vizualizar PROMPT "Intrari"
ON SELECTION BAR 1 OF vizualizar ;
    DO _1j10vft5n ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")
ON SELECTION BAR 2 OF vizualizar ;
    DO _1j10vft5o ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")
ON SELECTION BAR 3 OF vizualizar ;
    DO _1j10vft5p ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")

DEFINE POPUP actualizar MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF actualizar PROMPT "Adaugare"
DEFINE BAR 2 OF actualizar PROMPT "Modificare/Stergere"
ON SELECTION BAR 1 OF actualizar ;
    DO _1j10vft5q ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")
ON SELECTION BAR 2 OF actualizar ;
    DO _1j10vft5r ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")
```

```
DEFINE POPUP rapoarte MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF rapoarte PROMPT "Lista materiale/Data"
DEFINE BAR 2 OF rapoarte PROMPT "Lista Furnizori/Materiale"
ON SELECTION BAR 1 OF rapoarte do form mat_data
ON SELECTION BAR 2 OF rapoarte ;
    DO _1j10vft5s ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")


*       ********************************************************
*       *
*       * _1J10VFT5M  ON SELECTION PAD
*       *
*       * Procedure Origin:
*       *
*       * From Menu:  MENU1.MPR,            Record:   16
*       * Called By:  ON SELECTION PAD
*       * Prompt:     Iesire
*       * Snippet:    1
*       *
*       ********************************************************
    PROCEDURE _1j10vft5m
        Clear events
        CLOSE database
        SET sysmenu to default


        *       ********************************************************
        *       *
        *       * _1J10VFT5N  ON SELECTION BAR 1 OF POPUP vizualizar
        *       *
        *       * Procedure Origin:
        *       *
        *       * From Menu:  MENU1.MPR,            Record:    5
        *       * Called By:  ON SELECTION BAR 1 OF POPUP vizualizar
        *       * Prompt:     Materiale
        *       * Snippet:    2
        *       *
        *       ********************************************************
    PROCEDURE _1j10vft5n
```

```
    SELECT 2
    BROWSE fields cod_mat, denumire noedit


    *       ********************************************************
    *       *
    *       * _1J10VFT5O  ON SELECTION BAR 2 OF POPUP vizualizar
    *       *
    *       * Procedure Origin:
    *       *
    *       * From Menu:  MENU1.MPR,            Record:    6
    *       * Called By:  ON SELECTION BAR 2 OF POPUP vizualizar
    *       * Prompt:     Furnizori
    *       * Snippet:    3
    *       *
    *       ********************************************************
PROCEDURE _1j10vft5o
    SELECT 3
    BROWSE fields cod_furn, denumire noedit


    *       ********************************************************
    *       *
    *       * _1J10VFT5P  ON SELECTION BAR 3 OF POPUP vizualizar
    *       *
    *       * Procedure Origin:
    *       *
    *       * From Menu:  MENU1.MPR,            Record:    7
    *       * Called By:  ON SELECTION BAR 3 OF POPUP vizualizar
    *       * Prompt:     Intrari
    *       * Snippet:    4
    *       *
    *       ********************************************************
PROCEDURE _1j10vft5p
    SELECT 4
    BROWSE noedit


    *       ********************************************************
    *       *
    *       * _1J10VFT5Q  ON SELECTION BAR 1 OF POPUP actualizar
```

```
    *       *
    *       * Procedure Origin:
    *       *
    *       * From Menu:  MENU1.MPR,            Record:   10
    *       * Called By:  ON SELECTION BAR 1 OF POPUP actualizar
    *       * Prompt:     Adaugare
    *       * Snippet:    5
    *       *
    *       ********************************************************
PROCEDURE _1j10vft5q
    DO form adaug


    *       ********************************************************
    *       *
    *       * _1J10VFT5R  ON SELECTION BAR 2 OF POPUP actualizar
    *       *
    *       * Procedure Origin:
    *       *
    *       * From Menu:  MENU1.MPR,            Record:   11
    *       * Called By:  ON SELECTION BAR 2 OF POPUP actualizar
    *       * Prompt:     Modificare/Stergere
    *       * Snippet:    6
    *       *
    *       ********************************************************
PROCEDURE _1j10vft5r
    DO form modificare


    *       ********************************************************
    *       *
    *       * _1J10VFT5S  ON SELECTION BAR 2 OF POPUP rapoarte
    *       *
    *       * Procedure Origin:
    *       *
    *       * From Menu:  MENU1.MPR,            Record:   15
    *       * Called By:  ON SELECTION BAR 2 OF POPUP rapoarte
    *       * Prompt:     Lista Furnizori/Materiale
    *       * Snippet:    7
    *       *
    *       ********************************************************
```

```
    PROCEDURE _1j10vft5s
        SELECT 4
        SET order to tag cod_mat of materiale.cdx in materiale
        SET relation to cod_mat into materiale additive
        SET order to tag cod_furn of furnizori.cdx in furnizori
        SET relation to cod_furn into furnizori additive
        REPORT form furniz_mater preview
        SET relation off into materiale
        SET relation off into furnizori
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MODIFICARE.SCX
Name = "Dataenvironment"
Top = 0
Left = -1
Height = 294
Width = 424
DoCreate = .T.
Caption = "Modificare / Stergere"
Name = "Form1"
ErasePage = .T.
PageCount = 3
Top = 9
Left = 10
Width = 400
Height = 266
TabIndex = 1
Name = "Pageframe1"
Page1.FontBold = .T.
Page1.Caption = "Furnizori"
Page1.Name = "Page1"
Page2.FontBold = .T.
Page2.Caption = "Materiale"
Page2.Name = "Page2"
Page3.FontBold = .T.
Page3.Caption = "Intrari"
Page3.Name = "Page3"
PROCEDURE Page1.Activate
    SELECT 3
    GO top
    denf=''
    thisform.Pageframe1.Page1.refresh
```

```
    thisform.Pageframe1.Page1.text1.setfocus
ENDPROC
PROCEDURE Page2.Init
    PUBLIC umas(6)
    umas(1)='BUC'
    umas(2)='KG'
    umas(3)='TO'
    umas(4)='MC'
    umas(5)='MP'
    umas(6)='ML'


ENDPROC
PROCEDURE Page2.Activate
    SELECT 2
    GO top
    denm=''
    thisform.Pageframe1.Page2.refresh
    thisform.Pageframe1.Page2.text1.setfocus
ENDPROC
PROCEDURE Page3.Activate
    SELECT 2
    GO top
    SELECT 3
    GO top
    SELECT 4
    GO top
    thisform.Pageframe1.Page3.refresh
ENDPROC
ControlSource = "cod_furn"
Height = 21
Left = 92
TabIndex = 1
Top = 11
Width = 81
Name = "Text1"
ControlSource = "denumire"
Height = 23
Left = 92
TabIndex = 2
Top = 44
```

```
Width = 250
Name = "Text2"
ControlSource = "adresa"
Height = 38
Left = 92
TabIndex = 3
Top = 85
Width = 245
Name = "Text3"
ControlSource = "tel"
Height = 23
Left = 92
TabIndex = 4
Top = 133
Width = 113
Name = "Text4"
Top = 174
Left = 246
Height = 21
Width = 62
FontBold = .T.
Caption = "Stergere"
TabIndex = 5
ForeColor = 255,0,0
Name = "Command1"
PROCEDURE Click
    codf=cod_furn
    Delete
    PACK
    SELECT 4
    LOCATE for codf=cod_furn
    DO while found(4)
        Delete
        CONTINUE
    ENDDO
    PACK
    SELECT 3
    thisform.Pageframe1.Page1.refresh
    thisform.Pageframe1.Page1.text1.setfocus
ENDPROC
FontBold = .T.
```

```
Caption = "Cod furnizor"
Height = 21
Left = 3
Top = 11
Width = 72
TabIndex = 6
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 3
Top = 43
Width = 80
TabIndex = 7
Name = "Label2"
FontBold = .T.
Caption = "Adresa"
Height = 22
Left = 3
Top = 83
Width = 85
TabIndex = 8
Name = "Label3"
FontBold = .T.
Caption = "Telefon"
Height = 20
Left = 3
Top = 135
Width = 74
TabIndex = 9
Name = "Label4"
Top = 135
Left = 239
Height = 20
Width = 31
Picture = top.bmp
Caption = ""
Name = "Command2"
PROCEDURE Click
    GO top
    thisform.Pageframe1.Page1.refresh
```

```
ENDPROC
Top = 135
Left = 317
Height = 20
Width = 31
Picture = next.bmp
Caption = ""
Name = "Command3"
PROCEDURE Click
    Skip 1
    IF eof(3)
        GO bottom
        MESSAGEBOX('Sfârsit fisier !',64)
    ENDIF
    thisform.Pageframe1.Page1.refresh
ENDPROC
Top = 135
Left = 361
Height = 20
Width = 31
Picture = end.bmp
Caption = ""
Name = "Command4"
PROCEDURE Click
    GO bottom
    thisform.Pageframe1.Page1.refresh

ENDPROC
Top = 135
Left = 278
Height = 20
Width = 31
Picture = prev.bmp
Caption = ""
Name = "Command5"
PROCEDURE Click
    Skip -1
    IF bof(3)
        GO top
        MESSAGEBOX('Inceput fisier !',64)
    ENDIF
```

```
    thisform.Pageframe1.Page1.refresh
ENDPROC
ControlSource = "denf"
Height = 24
Left = 92
Top = 173
Width = 145
Name = "Text5"
PROCEDURE KeyPress
    LPARAMETERS nKeyCode, nShiftAltCtrl


ENDPROC
PROCEDURE Valid
    LOCATE for upper(left(denf,len(alltrim(denf))))=;
        upper(left(denumire,len(alltrim(denf))))
    IF found(3) then
        thisform.Pageframe1.Page1.refresh
    ELSE
        MESSAGEBOX('Nu exista !',64)
    ENDIF
ENDPROC
FontBold = .T.
Caption = "Cauta furnizor"
Height = 22
Left = 3
Top = 176
Width = 78
Name = "Label5"
ControlSource = "cod_mat"
Height = 25
Left = 115
TabIndex = 1
Top = 16
Width = 76
Name = "Text1"
ControlSource = "denumire"
Height = 25
Left = 115
TabIndex = 2
Top = 64
```

```
Width = 256
Name = "Text2"
FontBold = .T.
Caption = "Cod material"
Height = 24
Left = 10
Top = 20
Width = 82
TabIndex = 5
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 10
Top = 70
Width = 81
TabIndex = 6
Name = "Label2"
RowSourceType = 5
RowSource = "umas"
Enabled = .T.
Height = 71
Left = 115
TabIndex = 3
Top = 110
Width = 100
Name = "List1"
PROCEDURE Click
    WITH thisform.Pageframe1.Page2
        .Text4.value=.list1.value
        um=.Text4.value
        .Text4.refresh
        .Text3.setfocus
    ENDWITH

ENDPROC
FontBold = .T.
Caption = "Unitate de masura"
Height = 22
Left = 10
Top = 120
```

```
Width = 105
Name = "Label3"
ControlSource = "denm"
Height = 25
Left = 115
Top = 191
Width = 198
Name = "Text3"
PROCEDURE Valid
    LOCATE for upper(left(denm,len(alltrim(denm))))=;
        upper(left(denumire,len(alltrim(denm))))
    IF found(2) then
        thisform.Pageframe1.Page2.list1.value=um
        thisform.Pageframe1.Page2.refresh
    ELSE
        MESSAGEBOX('Nu exista !',64)
    ENDIF
ENDPROC
FontBold = .T.
Caption = "Cauta material"
Height = 24
Left = 10
Top = 197
Width = 102
Name = "Label4"
Top = 160
Left = 236
Height = 20
Width = 31
Picture = top.bmp
Caption = ""
Name = "Command2"
PROCEDURE Click
    GO top
    thisform.Pageframe1.Page2.refresh
ENDPROC
Top = 160
Left = 314
Height = 20
Width = 31
Picture = next.bmp
```

```
Caption = ""
Name = "Command3"
PROCEDURE Click
    Skip 1
    IF eof(2)
        GO bottom
        MESSAGEBOX('Sfârsit fisier !',64)
    ENDIF
    thisform.Pageframe1.Page2.refresh
ENDPROC
Top = 160
Left = 358
Height = 20
Width = 31
Picture = end.bmp
Caption = ""
Name = "Command4"
PROCEDURE Click
    GO bottom
    thisform.Pageframe1.Page2.refresh

ENDPROC
Top = 160
Left = 275
Height = 20
Width = 31
Picture = prev.bmp
Caption = ""
Name = "Command5"
PROCEDURE Click
    Skip -1
    IF bof(2)
        GO top
        MESSAGEBOX('Inceput fisier !',64)
    ENDIF
    thisform.Pageframe1.Page2.refresh
ENDPROC
FontBold = .T.
ControlSource = "um"
Enabled = .T.
Height = 24
```

```
Left = 18
ReadOnly = .T.
Top = 144
Width = 83
Name = "Text4"
FontBold = .T.
ControlSource = "cod_mat"
Height = 25
Left = 86
ReadOnly = .T.
TabIndex = 1
Top = 85
Width = 84
Name = "Text1"
FontBold = .T.
ControlSource = "cod_furn"
Height = 25
Left = 85
ReadOnly = .T.
TabIndex = 3
Top = 204
Width = 82
Name = "Text2"
ControlSource = "cant"
Height = 25
Left = 276
TabIndex = 5
Top = 30
Width = 84
Name = "Text3"
ControlSource = "pu"
Height = 25
Left = 276
TabIndex = 6
Top = 67
Width = 107
Name = "Text4"
ControlSource = "data_in"
Height = 25
Left = 276
TabIndex = 7
```

```
Top = 102
Width = 75
Name = "Text5"
BoundColumn = 1
RowSourceType = 6
RowSource = "materiale"
ControlSource = "denumire"
Height = 51
Left = 5
Sorted = .F.
TabIndex = 2
Top = 28
Width = 164
BoundTo = .T.
Name = "List1"
PROCEDURE Click
    SELECT 2
    codm=cod_mat
    thisform.Pageframe1.Page3.text1.refresh
ENDPROC
RowSourceType = 6
RowSource = "furnizori"
ControlSource = "denumire"
Height = 52
Left = 4
Sorted = .F.
TabIndex = 4
Top = 142
Width = 163
Name = "List2"
PROCEDURE Click
    SELECT 3
    codf=cod_furn
    thisform.Pageframe1.Page3.Text2.refresh
    thisform.Pageframe1.Page3.Text3.setfocus
ENDPROC
FontBold = .T.
Caption = "Cod material"
Height = 23
Left = 7
Top = 87
```

```
Width = 81
TabIndex = 9
Name = "Label1"
FontBold = .T.
Caption = "Cod furnizor"
Height = 24
Left = 8
Top = 207
Width = 69
TabIndex = 10
Name = "Label2"
FontBold = .T.
Caption = "Cantitate"
Height = 28
Left = 204
Top = 34
Width = 57
TabIndex = 11
Name = "Label3"
FontBold = .T.
Caption = "Pret unitar"
Height = 28
Left = 204
Top = 72
Width = 66
TabIndex = 12
Name = "Label4"
FontBold = .T.
Caption = "Data intrarii"
Height = 26
Left = 204
Top = 105
Width = 69
TabIndex = 13
Name = "Label5"
FontBold = .T.
Caption = "Denumire material"
Height = 16
Left = 14
Top = 7
Width = 125
```

```
TabIndex = 14
ForeColor = 0,64,0
Name = "Label6"
FontBold = .T.
Caption = "Denumire furnizor"
Height = 22
Left = 7
Top = 122
Width = 142
TabIndex = 15
ForeColor = 255,0,0
Name = "Label7"
Top = 156
Left = 213
Height = 20
Width = 31
Picture = top.bmp
Caption = ""
Name = "Command2"
PROCEDURE Click
    SELECT 4
    GO top
    thisform.Pageframe1.Page3.refresh
ENDPROC
Top = 156
Left = 291
Height = 20
Width = 31
Picture = next.bmp
Caption = ""
Name = "Command3"
PROCEDURE Click
    SELECT 4
    Skip 1
    IF eof(4)
        GO bottom
        MESSAGEBOX('Sfârsit fisier !',64)
    ENDIF
    thisform.Pageframe1.Page3.refresh
    SELECT 2
    LOCATE for intrari.cod_mat=materiale.cod_mat
```

```
    IF found(2)
        thisform.Pageframe1.Page3.list1.value=denumire
    ENDIF
    SELECT 4
    thisform.Pageframe1.Page3.refresh

ENDPROC
Top = 156
Left = 335
Height = 20
Width = 31
Picture = end.bmp
Caption = ""
Name = "Command4"
PROCEDURE Click
    SELECT 4
    GO bottom
    thisform.Pageframe1.Page3.refresh

ENDPROC
Top = 156
Left = 252
Height = 20
Width = 31
Picture = prev.bmp
Caption = ""
Name = "Command5"
PROCEDURE Click
    SELECT 4
    Skip -1
    IF bof(4)
        GO top
        MESSAGEBOX('Inceput fisier !',64)
    ENDIF
    thisform.Pageframe1.Page3.refresh
    SELECT 3
    LOCATE for intrari.cod_furn=furnizori.cod_furn
    IF found(3)
        thisform.Pageframe1.Page3.List2.value=denumire
    ENDIF
    SELECT 4
```

```
    thisform.Pageframe1.Page3.refresh
ENDPROC
Top = 195
Left = 222
Height = 25
Width = 63
FontBold = .T.
Caption = "Stergere"
ForeColor = 255,0,0
Name = "Command1"
PROCEDURE Click
    btnvalue=0
    btnvalue=messagebox('Stergere ?',4+32+256)
    IF btnvalue=6 then
        Delete
        PACK
    ENDIF
    thisform.Pageframe1.Page3.refresh
ENDPROC
Top = 236
Left = 345
Height = 25
Width = 49
FontBold = .T.
Caption = "Iesire"
Name = "Command2"
PROCEDURE Click
    thisform.release
ENDPROC
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\NEXT.BMP
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\PREV.BMP
****** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\TOP.BMP
```

12.2.2. Realizarea documentației şi a aplicației în format executabil

Pentru obținerea documentației referitoare la sursele proiectului se parcurg următorii paşi:

a) Se închide aplicația **Project Manager**;

b) Din meniul **Tools** se alege opțiunea **Wizards** / **Documenting**. Aplicația **Documenting Wizard** parcurge 6 paşi pentru elaborarea documentației proiectului:
   1. Se alege fişierul sursă (proiectul *magazie.pjx*) inclusiv calea către acesta. Se apasă butonul **Next**;
   2. Se setează caractere mari pentru cuvinte cheie (**Keywords**) şi simboluri;
   3. Se setează identare text (**Tab** sau **Space**);
   4. Se adăugă antet;
   5. Se selectează tipuri de rapoarte;
   6. Se încheie cu specificarea subdirectorului în care vor fi generate rapoartele (existent sau nou creat).

Fişierul MAGAZIE.LST va conține codul sursă (fişier de tip text care poate fi ulterior trecut într-un fişier Word) pentru toate programele şi procedurile ataşate obiectelor componentelor folosite în proiect.

Se generează un tabel care va conține toate toate fişierele proiectului, generate, primare şi implicite (videoformate, rapoarte, meniuri), fişierele de tip *BMP* etc.

De exemplu, videoformatele (**Forms**) sunt salvate într-un fişier cu extensia *.scx.* La crearea videoformatului, tabelul cu extensia *.scx* conține o înregistrare pentru videoformat, o înregistrare pentru datele de mediu şi două înregistrări pentru uz intern. Pentru fiecare obiect adăugat în videoformat se adaugă o nouă înregistrare.

Anumite componente ale Visual FoxPro constau din fişiere multiple: un fişier primar şi unul sau mai multe fişiere implicite.

De exemplu, când se crează un videoformat, Visual FoxPro crează un fişier *.scx* (fişier primar) şi un fişer *.sct* (fişier implicit)

Următoarele componente au fişiere multiple (Tabelul 1)

Tabelul 1

Componentă	**Fişiere primare**	**Fişiere implicite**
Form	.scx	.sct
Report	.frx	.frt

Label	.lbx	.lbt
Class Library	.vcx	.vct
Menu	.mnx	.mnt
Table	.dbf	.fpt, .cdx, .idx
Database	.dbc	.dct, .dcx

Tabelul 2 prezintă extensiile şi tipurile de fişiere asociate care sunt utilizate în Visual FoxPro

Tabel 2

Extensie	**Tip fişier**
.act	Documenting Wizard action diagram
.app	Generated application or Active Document
.cdx	Compound index
.chm	Compiled HTML Help
.dbc	Database
.dbf	Table
.dbg	Configurație debugger
.dct	memo database
.dcx	index database
.dep	Fişier dependență (creat de Setup Wizard)
.dll	Windows Dynamic Link Library
.err	Eroare de compilare
.esl	Visual FoxPro suport bibliotecă
.exe	Program executabil
.fky	Macro
.fll	Visual FoxPro Dynamic Link Library
.fmt	Format Fişier
.fpt	Table memo
.frt	Report memo
.frx	Report

.fxp	Program compilat
.h	Antet fişier (pentru includere în Visual FoxPro or C/C++ program)
.hlp	WinHelp
.htm	HTML
.idx	Index, compact index
.lbt	Label memo
.lbx	Label
.log	Coverage log
.lst	Documenting Wizard list
.mem	Salvare variabile
.mnt	Menu memo
.mnx	Menu
.mpr	Program menu generat
.mpx	Program menu compilat
.ocx	ActiveX control
.pjt	Project memo
.pjx	Project
.prg	Program
.qpr	Program query generat
.qpx	Program query compilat
.sct	Form memo
.scx	Form
.spr	Screen Program generat (numai pentru versiunile anteriaore FoxPro)
.spx	Screen Program compilat (numai pentru versiunile anteriaore FoxPro)
.tbk	Memo backup
.txt	Text
.vct	Visual class library memo

.vcx	Clasă de biblioteci Visual
.vue	FoxPro 2.*x* view
.win	Fişier Window

Pentru a genera programul executabil al aplicației, se deschide **Project Manager** pentru aplicația *magazie.pjx* şi se apasă butonul **Build** şi se specifică opțiunile pentru construire, Figura 1.

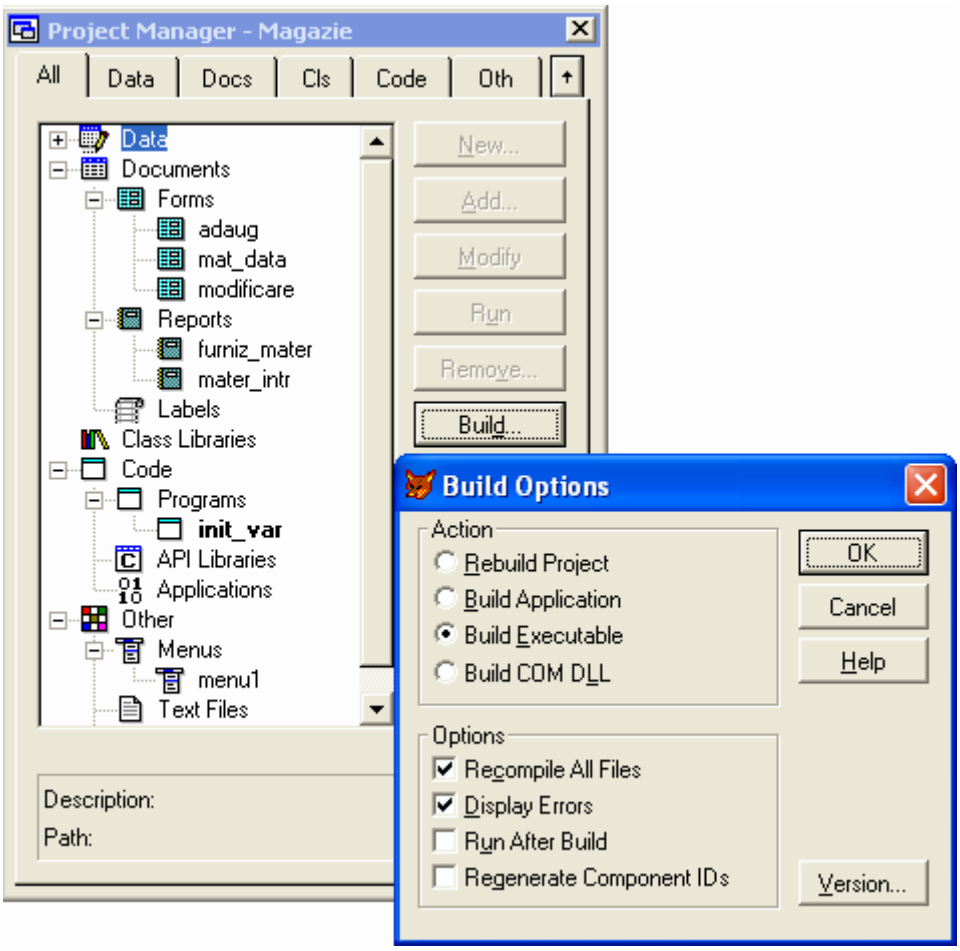


Fig. 1

Opțiunile de construire se refereră la următoarele acțiuni:

- reconstruirea proiectului (**Rebuild Project**);
- construirea aplicației (**Build Application**);
- construirea programului executabil (**Build Executable**);
- construirea componentelor COM (**Component Object Model** pentru includerea controalelor **ActivX** sau dacă s-a creat un **Automation server**) sub formă de DLL (librărie de legătură dinamică).

Acțiune de construire a executabilului are un subset de opțiuni:

- recompilarea tuturor fişierelor;
- afişarea erorilor;
- lansare în execuție după construire;
- regenerarea identificării obiectelor.

Pentru celelalte acțiuni numărul opțiunilor din subset poate varia în funcție de acțiune. De exemplu, pentru construirea elementelor COM vor fi active doar trei opțiuni, fără cea de rulare după construire (un DLL nu poate fi lansat în execuție fiind o librărie dinamică).

Programul executabil generat poate fi rulat pe acelaşi calculator sau pe un altul, cu condiția ca acesta să aibă Visual FoxPro instalat.

12.2.3. Utilizarea Setup Wizard

Pentru a mări portabilitatea aplicației se foloseşte **Setup Wizard**, în vederea realizării de subrutine de instalare a aplicației pe un alt calculator.

Din meniul **Tools** se alege submeniul **Wizard**, apoi opțiunea **Setup**. Se parcurg 7 paşi:

1. Localizarea fişierelor – se specifică subdirectorul care conține fişierele şi subdirectoarele ce trebuie distribuite. Nu se poate folosi **Distrib** ca nume de subdirector în care se află fişierele şi subdirectoarele ce se vor distribui;
2. Specificarea componentelor care vor fi incluse în pachetul de distribuție, şi anume:

- **Visual FoxPro 6.0 Runtime** – librăriile Visual FoxPro;
- **Microsoft 8.0 Graph Runtime**;
- **ODBC Drivers**;
- **COM Components**;
- **ActiveX Controls**;
- **HTML help engine**;

3. Crearea subdirectorului pentru **Disk Image**:

- **1.44 MB 3.5** – wizard-ul crează imagini pentru floppy disk;
- **Websetup** (compressed) – wizard-ul crează un program de instalare dens compresat proiectat pentru descărcare rapidă de pe un web site;
- **Netsetup** – wizard-ul crează un singur director care va conține toate fişierele.

4. Specificarea opțiunilor de instalare – referitoare la:

- **Setup dialog box caption** – titlul ferestrei de **Setup,** opțional;

- **Copyright information** – obligatoriu;
- **Post-setup executable** – specificarea unei acțiuni post instalare, opțional;

5. Identificarea destinației implicite pentru fişier – rutina setup va plasa aplicația în directorul care se specifică în **Default directory** . Opțiunea **Program group** va crea un grup de programe pentru aplicația instalată care se va regăsi în meniul **Start** al calculatorului utilizatorului. Opțiunea **User can modify**, oferă posibilitatea modificării subdirectorului implicit de instalatre;
6. Schimbarea setărilor fişierelor – se afişează fişierele şi opțiunile făcute pentru ele cu posibilitatea de a se opera modificări legate de nume, destinație sau alte modificări;
7. Terminarea procesului **Setup Wizard** – la acest pas se poate crea un fişier de dependență (.dep) care permite folosirea altor utilități la instalarea aplicației. După apăsarea butonului de **Finish, Setup Wizard** realizează următoare acțiuni:

- înregistrează configurația pentru următoarea utilizare a distribuției din aceeaşi cale;
- lansează procesul de creare a imaginilor disc.

Imaginile disc pot fi copiate pe orice mediu de stocare şi poate fi distribuit utilizatorilor aplicației.

A. Anexa
Limite ale SISTEMULUI VFP 6.0

Caracteristică	Valoare
Nr. maxim de înregistrări într-o tabelă	1 miliard
Mărimea maximă a unei tabele	2 Gb
Nr. maxim de caractere pe înregistrare	65.500
Nr. maxim de câmpuri pe înregistrare	255
Nr. maxim de tabele deschise simultan	255
Nr. maxim de caractere pentru un câmp	254
Nr. maxim de bytes pentru cheia de index compus	100
Nr. maxim de bytes pentru cheia de index simplu	240
Nr. maxim de fişiere de index deschise pentru o tabelă	nelimitat
Nr. maxim de fişiere de index deschise pentru toate zonele de lucru	nelimitat
Nr. maxim de legături	nelimitat
Lungimea maximă a unei expresii relaționale	nelimitat
Valoarea maximă pentru întregi	2.147.483.647
Precizia în calculul numeric	16
Mărimea maximă pentru un câmp de tip caracter	254
Mărimea maximă pentru un câmp de tip numeric	20
Nr. maxim de caractere pentru un câmp dintr-o tabelă liberă	10
Nr. maxim de caractere pentru un câmp dintr-o tabelă care aparține unei BD	128
Nr. maxim de variabile	65.000
Nr. maxim de masive	65.000
Nr. maxim de elemente într-un masiv	65.000
Nr. maxim de linii sursă într-un program	nelimitat
Mărimea maximă a modulului de program compilat	64 Kb
Nr. maxim de proceduri (subprograme)	nelimitat
Nr. maxim de apeluri DO imbricate	128
Nr. maxim de comenzi de ciclare	384
Nr. maxim de parametri transmişi	27
Nr. maxim de tranzacții	5
Nr. maxim de obiecte la definirea unui raport	nelimitat

Nr. maxim de niveluri de grupare într-un raport	128
Nr. maxim de ferestre deschise	nelimitat
Nr. maxim de ferestre BROWSE deschise	255
Nr. maxim de caractere într-un şir de caractere	16.777.184
Nr. maxim de caractere într-o linie de comandă	8.192
Nr. maxim de fişiere deschise	limită sistem
Nr. maxim de câmpuri selectate prin comanda SELECT-SQL	255

Index de termeni

accesul la date
direct, 97
secvential, 97
actualizarea bazei de date, 13
algebră relațională
diferență, 20
diviziunea, 21
intersecția, 21
joncțiune, 21
joncțiune exterioară, 21
joncțiune exterioară completă, 21
joncțiune exterioară dreapta, 21
joncțiune exterioară stânga, 21
joncțiune naturală, 21
produs cartezian, 20
proiecția, 20
reuniunea, 20
selecția, 20
anularea marcării pentru ştergere fizică, 49
apelul unui (sub)program, 130
arhitectura sistemului de baze de date, 6
atribut, 4
complex, 29
cu o singură valoare, 29
cu set de valori, 29
derivat, 29
fără valoare, 29
bănci de date, 5
bază de date, 4
bază de date relațională, 28
câmp, 4
crearea etichetelor, 69
crearea interogărilor, 70
crearea rapoartelor, 68
crearea structurii unei tabele, 149
crearea tabelelor, 66
crearea videoformatelor, 67
de date
relațional, 8
deschiderea bazei de date, 52, 60
deschiderea unei tabele, 49
dictionar de date, 4
domeniu, 19
entitate, 4
fişier index, 94
compus, 94
simplu, 94
funcție utilizator, 129
închiderea bazei de date/tabelei, 64
Independența datelor
fizică, 10
logică, 10
indexarea tabelei, 92
înregistrare, 4
fizică, 4
logică, 4
interclasarea documentelor, 71
interogarea bazei de date, 13
legătura între tabele, 113
matrice, 85
metadate, 4
model de date, 6, 7
distribuit, 8
ierarhic, 8
in rețea, 8
primitiv, 8
semantic (orientat obiect), 9
modificarea înregistrărilor, 106
modificarea structurii, 91
bazei de date, 62
tabelei, 62
procedură utilizator, 129
programare
modulară, 119
orientată pe obiecte, 37
procedurală, 119
structurată, 119
programarea orientată pe obiecte
abstractizarea, 42
clasa, 37
clasa parinte, 38
evenimentul, 39
încapsularea, 41
mesaj, 38
metoda, 38
moştenirea, 41
obiect, 38
polimorfism, 41
proprietatea, 39
restricții de integritate, 38
subclasa, 37
redenumire fişier, 117
redundanță minimă, 14
relație, 19

reorganizarea bazei de date, 14
Schema în modelul relațional, 29
securitatea datelor, 14
sistem de gestiune a bazelor de date, 5, 9
  complet relațional, 36
  minimal relațional, 36
  relaționale, 19
sisteme de baze de date, 5
sortarea tabelei, 92
ştergerea fişierelor, 110
ştergerea înregistrărilor, 109
  fizică, 50, 63, 110, 144
  logică, 49, 63, 109, 144
structura bazei de date, 6
structura unei functii, 130
structura unei funcții, 130
structura unei proceduri, 129
subprograme imbricate, 130
tabelă, 19
transmiterea parametrilor, 130
  prin referință, 131
  prin valoare, 131
validarea câmpurilor, 55
variabilă
  globală, 85
  locală, 85
vectori, 85
zonă de lucru, 52

Bibliografie

1. Dima, G., Dima M. *FoxPro*, Editura Teora, Bucureşti, 1993.

2. Dima, G., Dima M. *FoxPro Meniuri*, Editura Teora, Bucureşti, 1994.

3. Kifer, M., Bernstein, A., Lewis, Ph. *Database Systems An Application-Oriented Approach*, State University of New York, Stony Brook, 2005.

4. Lungu, I. şi al. *Sistemul FoxPro Prezentare şi Aplicaţii*, Editura All, Bucureşti, 1993.

5. M. Velicanu, şi al. *Sisteme de Gestiune a Bazelor de Date*, Editura Petrion, Bucureşti, 2000;

6. Velicanu, M., Lungu, I., Muntean, M. *Dezvoltarea Aplicaţiilor cu Baze de Date în Visual FoxPro*, Editura All, Bucureşti, 2001.

7. *** Visual FoxPro 6.0 Manualele Microsoft

8. *** MSDN Library, April 2003.